Technical Exchange: RTI 2.0 Design

Mr. Steve Bachinsky, SAIC

9 October 1997

RTI 2.0 Design Overview

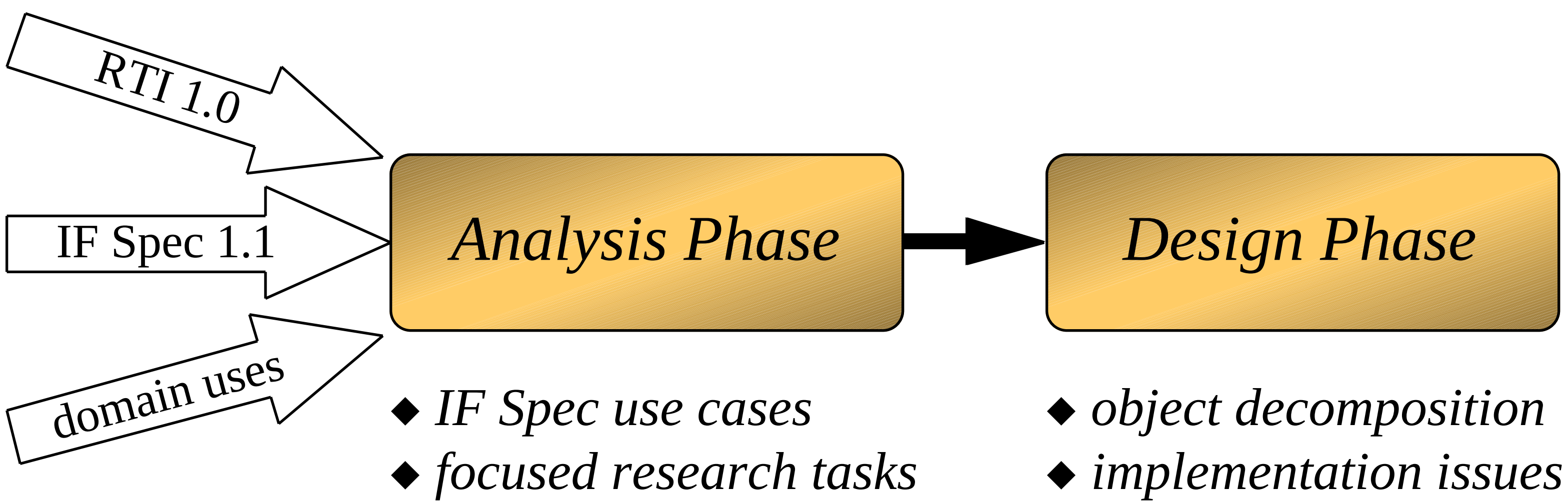
- Program Overview
- Analysis Phase
- Design Phase
- RTI 2.0 Design

Program Overview

- RTI 2.0 Phase I
  - four month effort involving analysis and design
- Design Objectives
  - provide full functionality (Interface Specification 1.1) with emphasis on performance across the M&S domains
  - develop a quality design that is buildable and maintainable
  - ensure support for a wide range of platforms, operating systems, and programming languages
- Test Plan
  - address development testing issues

RTI 2.0 Design Approach

- **Requirements**
  - **derived from Interface Specification and RTI 1.0 implementation**
  - **considered use patterns from federations representative of key M&S domains (training, analytic, hard real-time)**
  - **performed use cases for analysis of requirements**
- **Program Phases**
  - **divided into analysis and design phases**



RTI 2.0 Design Overview

- **Program Overview**
- **Analysis Phase**
  - **Interface Specification Use Cases**
  - **Focused Research Tasks**
- **Design Phase**
- **RTI 2.0 Design**

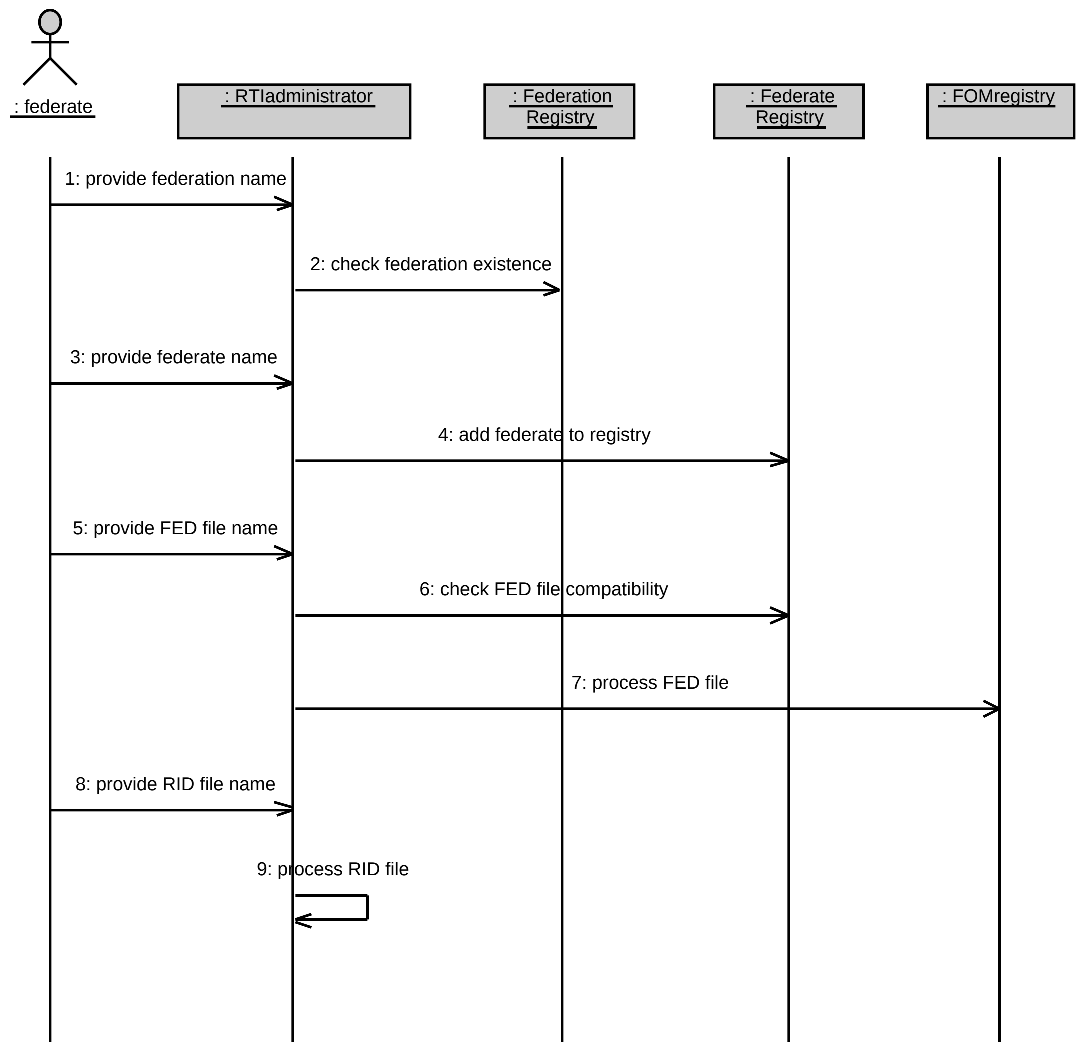
Analysis Phase

- Object-Oriented Analysis (OOA)
  - examine domain requirements
  - discover fundamental objects and relationships
- Use Cases
  - examines response of a system to a particular stimulus
  - captures detailed system requirements and identifies objects
- Critical Architectural Elements
  - performed focused research into the “long poles” of the design
  - examined implementation/portability issues and identified alternatives to support flexibility and extensibility

Interface Specification Use Cases

- Requirements Analysis
  - understand details associated with service operations
  - identify implementation issues with the Interface Specification
- Functional Decomposition
  - determine “functional objects” and actions which contribute to the system design
- CASE Tool Support
  - graphically depict the objects and actions required in performing a particular use case scenario
  - Rational Rose integrates “use case” and “logical” views
    - logical view captures design classes, associations, …

Join Federation Use Case



Focused Research Tasks

- LBTS Algorithm
- Threading Model & Event Scheduler
- Primary Data Flow
- Administrative Communications
- Data Addressing and Routing
- Network Abstraction
- Reliable Multicast

LBTS Algorithm

- Issue
  - delivery of time stamped order (TSO) messages requires the distributed computation of the lower bound on time stamp (LBTS)
- Requirements
  - performance
  - robustness
  - dynamic time regulating topology
- Strategies
  - conservative synchronization algorithms
  - global virtual time algorithms
  - distributed snapshot algorithms

Threading Model & Event Scheduler

- **Issue**
  - support different threading models and exploit multi-processor hardware
  - mechanism to facilitate event unification (RTI & federate)
- **Requirements**
  - federate configurable
  - platform portability
- **Strategies**
  - functional vs transactional threading decomposition
  - single threaded, separate RTI thread, reentrant federate
  - “Reactor” pattern
  - federate polling vs blocking (i.e., file descriptor)

Primary Data Flow

- **Issue**
  - identify primary data flow paths and minimize processing cycles per transaction
- **Requirements**
  - performance
  - validation of pre-conditions for Interface Specification services
- **Strategies**
  - conditional logic vs dynamic binding (“State” pattern)
  - pre-allocated memory pools
  - minimize data copy

Administrative Communications

- Issue
  - facilitate communication for administrative services
- Requirements
  - support point-to-point topologies and point-to-multipoint mechanisms
- Strategies
  - “Proxy/Server” pattern abstracts remote operation invocation
  - leverage existing RTI communications infrastructure

Data Addressing and Routing

- **Issue**
  - **configurable mechanism which minimizes the communication of unwanted data**
- **Requirements**
  - **support class (DM) and value based (DDM) filtering**
  - **scalability**
- **Strategies**
  - **multicast technology**
  - **static and dynamic gridding for DDM**
  - **dynamic producer/consumer mapping**

Network Abstraction

- **Issue**
  - isolate internal RTI modules from platform dependencies associated with I/O mechanisms
- **Requirements**
  - platform portability
  - performance
- **Strategies**
  - abstract transport mechanism behind standard interface
  - exploit operating system mechanisms for optimal performance

Reliable Multicast

- **Issue**
  - reliable multipoint-to-multipoint communication mechanism
- **Requirements**
  - scalability
  - performance
  - encapsulated within the virtual network module
- **Strategies**
  - ACK vs NACK based
  - centralized sequencer
  - RMP (COTS package with tailorable protocol)

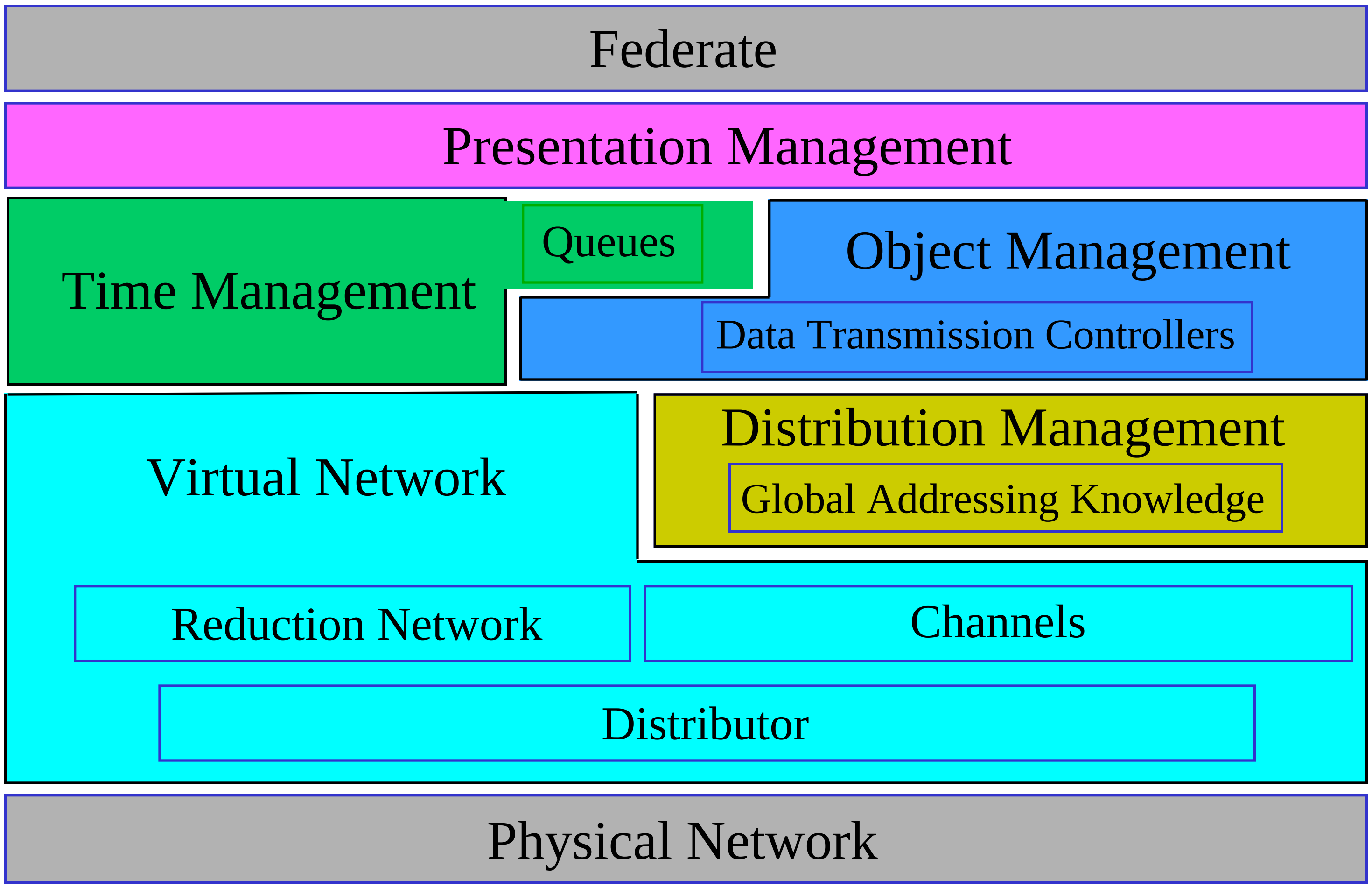
RTI 2.0 Final Design Review

- Program Overview
- Analysis Phase
- °Design Phase
    - -activities
    - -layered architecture
    - -design modules
- RTI 2.0 Design

Design Phase

- **Object-Oriented Design (OOD)**
  - refined objects and relationships from the analysis effort
  - examined architecture and implementation issues
- **Design Modules**
  - architecture separated into individual modules (or packages) which provide specific functionality and attempt to maximize decoupling
- **Class Diagrams**
  - identified class interfaces and relationships between classes
- **Collaboration Diagrams**
  - performed localized use cases to refine class functionality and relationships

RTI 2.0 Design Architecture



Design Modules

- **Presentation Manager**
  - provides the HLA compliant runtime interface between the RTI application library and the federate application
  - handles different programming languages and threading models
- **Administration**
  - general administrative support for federation executions
  - provides distributed operating system support including distributed object communication graphs
- **Process Model**
  - contains elements for managing events, timeouts, and processing threads

Design Modules

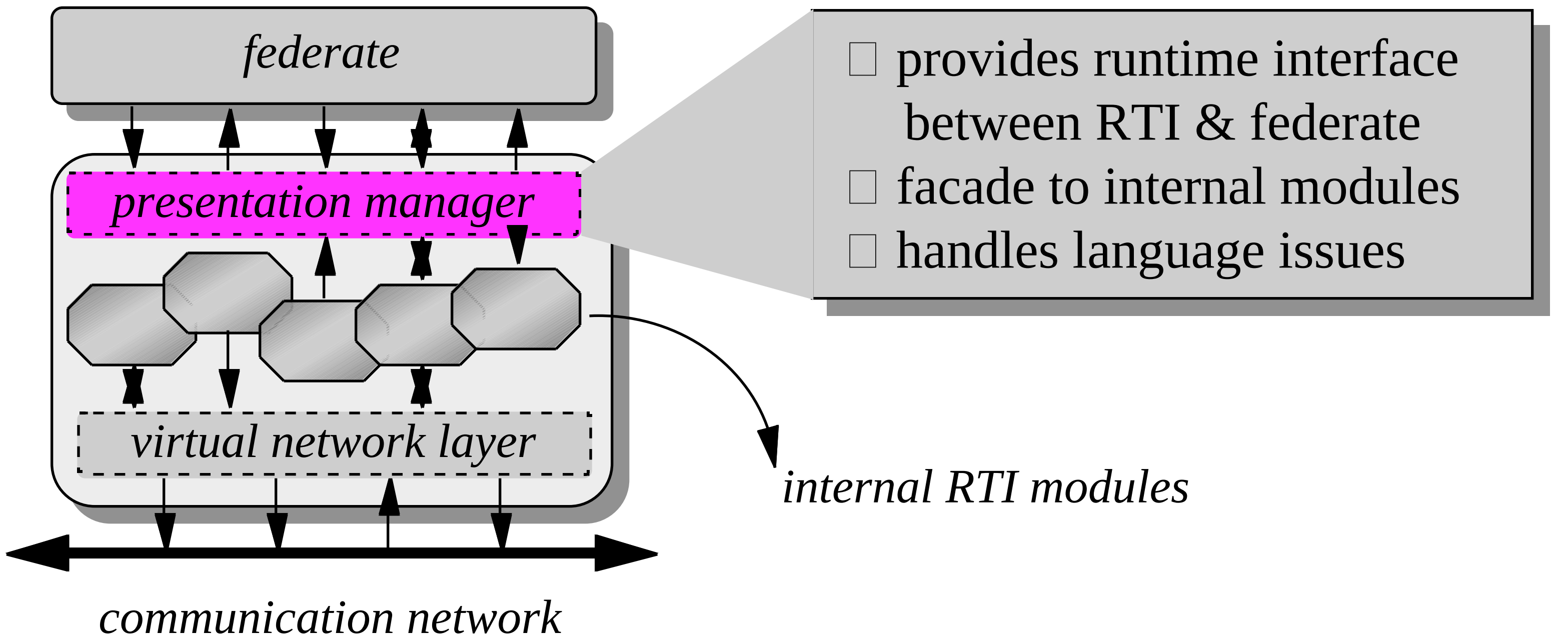
- Virtual Network
  - standard interface used to communicate with the network
  - hides communication mechanisms from the other modules
- Object Management
  - maintains information on federation objects and interactions
  - supports efficient data transfer between the federate and network
- Distribution Management
  - addresses data for transmission and reception purposes according to federation routing guidelines
- Time Management
  - controls the advancement of a federate’s time
  - correctly orders and releases data to the federate

RTI 2.0 Design Modules

°Presentation Manager
- Administration
- Process Model
- Virtual Network
- Object Management
- Distribution Management
- Time Management

Presentation Management Module

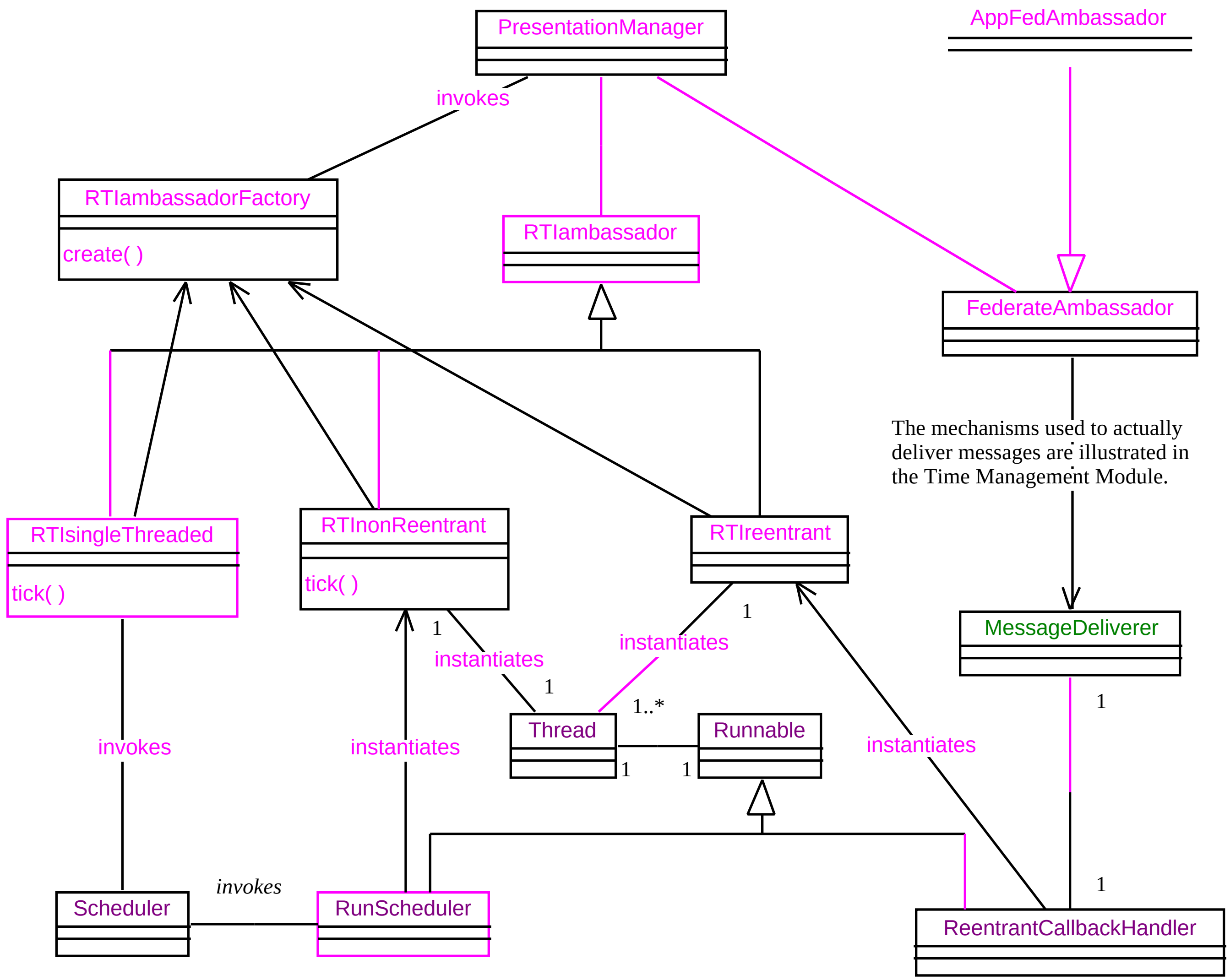
- **Language independent interface between RTI and federate**
- **Two Key Classes**
  - **RTIambassador**
  - **FederateAmbassador**



Presentation Manager Design Issues

- Supports language neutral interface to federate
  - multiple implementations are created for conversion with different federate programming languages
  - compliant with the Interface Specification
- Message Processing
  - receives message from internal modules which result in invocations on the FederateAmbassador
  - RTIambassador invocations result in the creation of messages which are dispatched to the appropriate internal module
- Provides configurability for threading model
  - RTIambassador subclassed and key methods overridden depending on RTI servicing requirements

PresentationManager Class Diagram

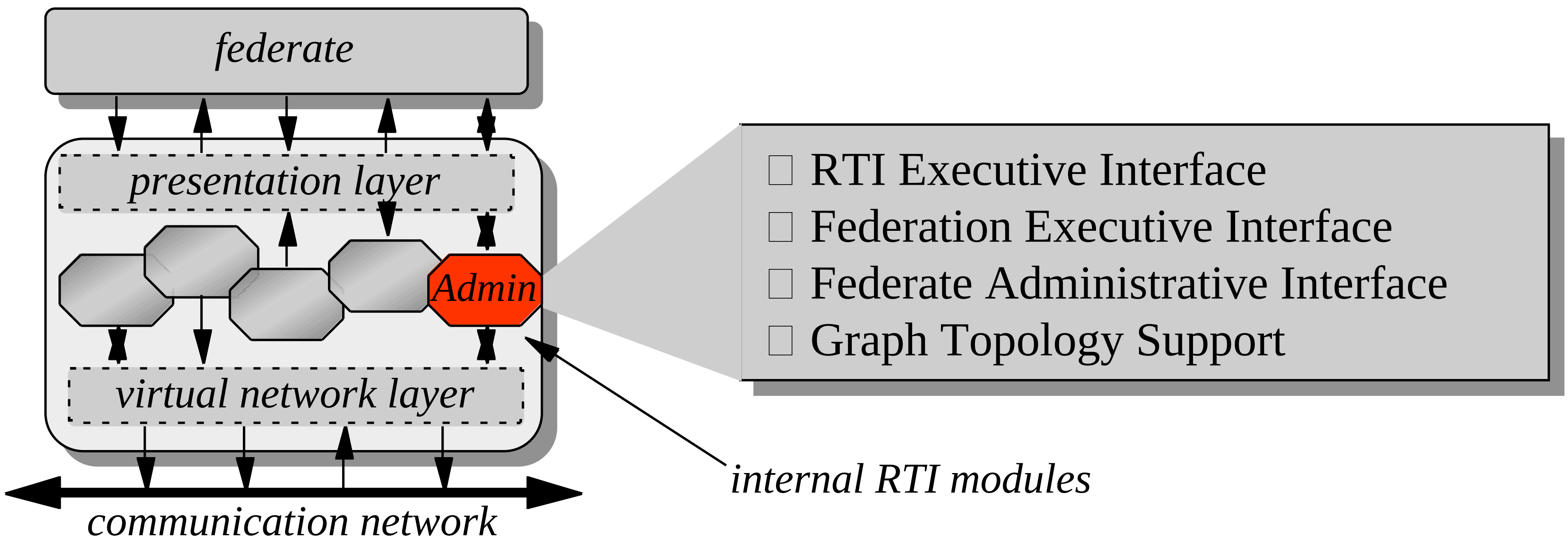


RTI 2.0 Design Modules

- Presentation Manager
- Administration
- Process Model
- Virtual Network
- Object Management
- Distribution Management
- Time Management

Administration Module

- **Administration Module Requirements**
  - **communications support for RTI to RTI administrative data**
  - **general distributed object support**
  - **graph topology support**



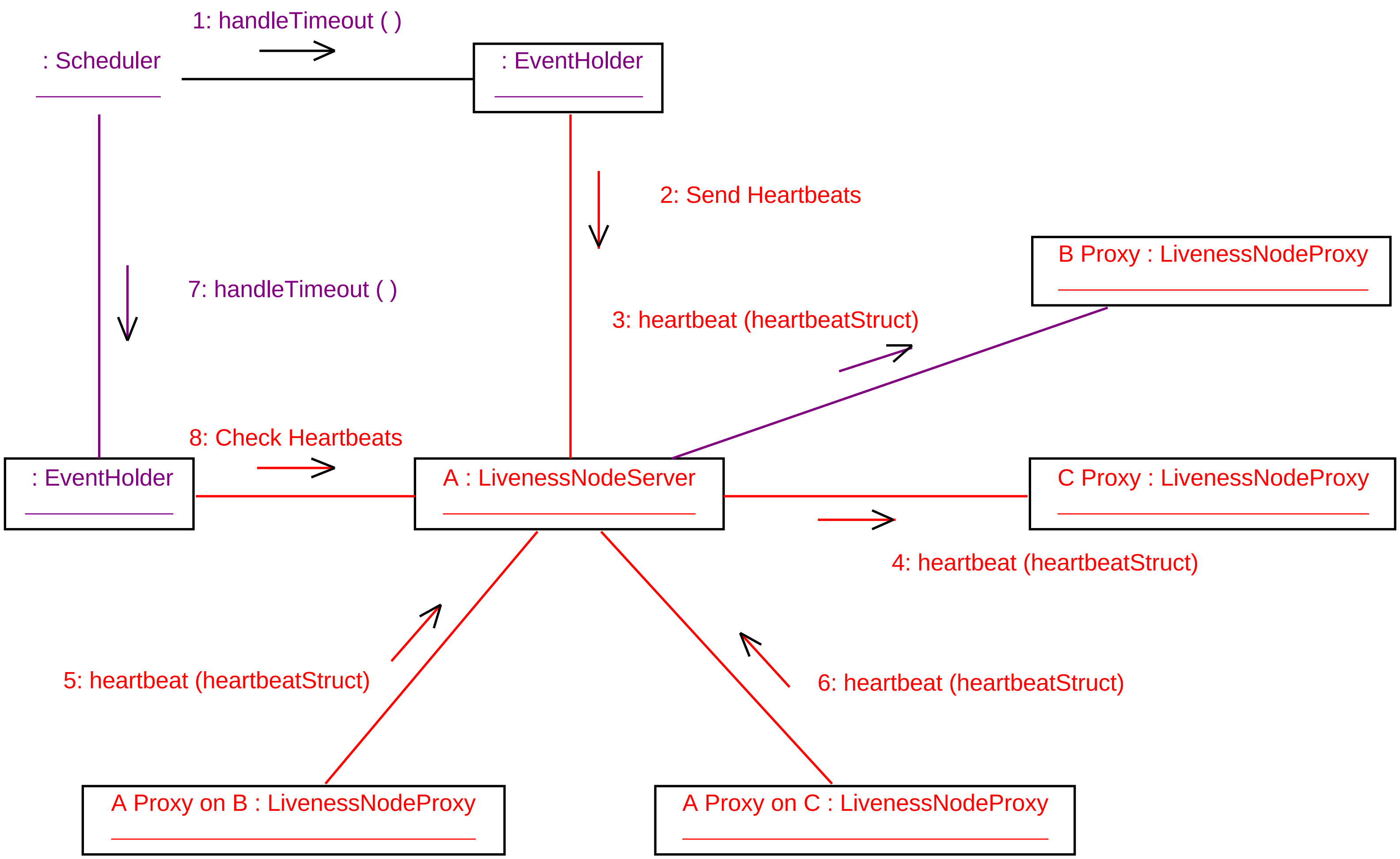
Key Module Functionality

- **RTI Executive**
  - **support for federation execution creation and resource allocation**
- **Federation Executive**
  - **central coordinator for federation administrative tasks**
- **Federate Administrator**
  - **provides administrative services for internal modules as well as other distributed RTI components**
- **Distributed Object Support**
  - **enables distributed communication using the “Proxy” pattern**
- **Graph Topology Support**
  - **provides functionality for local connectivity graphs**

Graph Topology Support

- **Support for general graph (mesh) topologies where the nodes are federates and the arc's are point-to-point links**
- **Reduction network support (used for LBTS calculations) is provided by a hierarchical graph**
- **Two types of servers**
  - **<u>graph nodes</u> - typically in federates, provide information**
  - **<u>graph managers</u> - typically in the federation executive, manages the graph topology, initiates calculations, etc.**
- ***<u>Proxies are distributed as operation parameters</u>***
  - **key to supporting topology distribution**

Liveness Heartbeat Collaboration Diagram



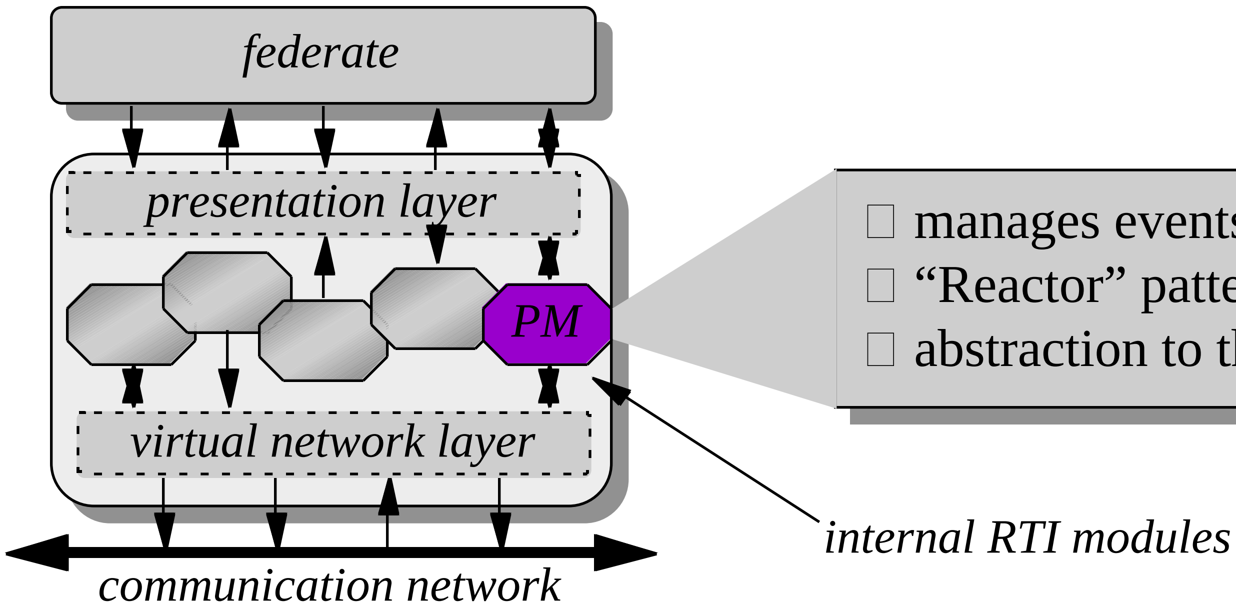
Local Operations for Machine "A"

RTI 2.0 Design Modules

- Presentation Manager
- Administration
- Process Model
- Virtual Network
- Object Management
- Distribution Management
- Time Management

Process Model Module

- **Process Model Module Requirements**
  - **event unification**
  - **transaction scheduling**
  - **support diverse threading schemes**



- manages events, timeouts, & threads
- "Reactor" pattern used for scheduling
- abstraction to threads

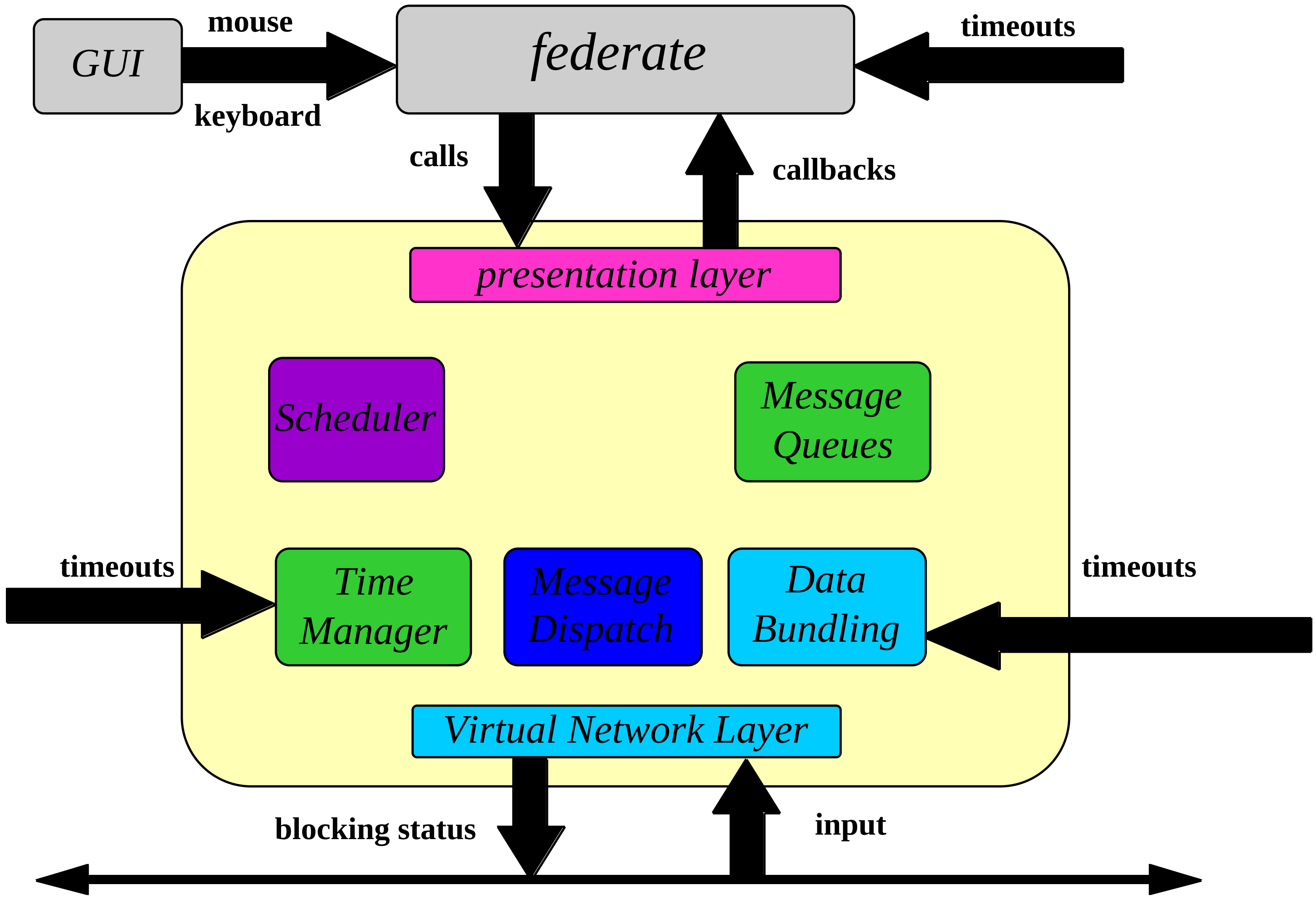
Process Model Design Issues

- Performance vs Ease of Use
  - managing events is straightforward but requires a framework for the diverse range of event types, implementing priorities, etc.
  - threading simplifies the developers task but can have significant performance implications which must be evaluated
- Portability
  - diverse platforms and operating systems may have differences in low level I/O APIs, and threading implementations
- Configurability
  - federate polling vs blocking (i.e., file descriptor)

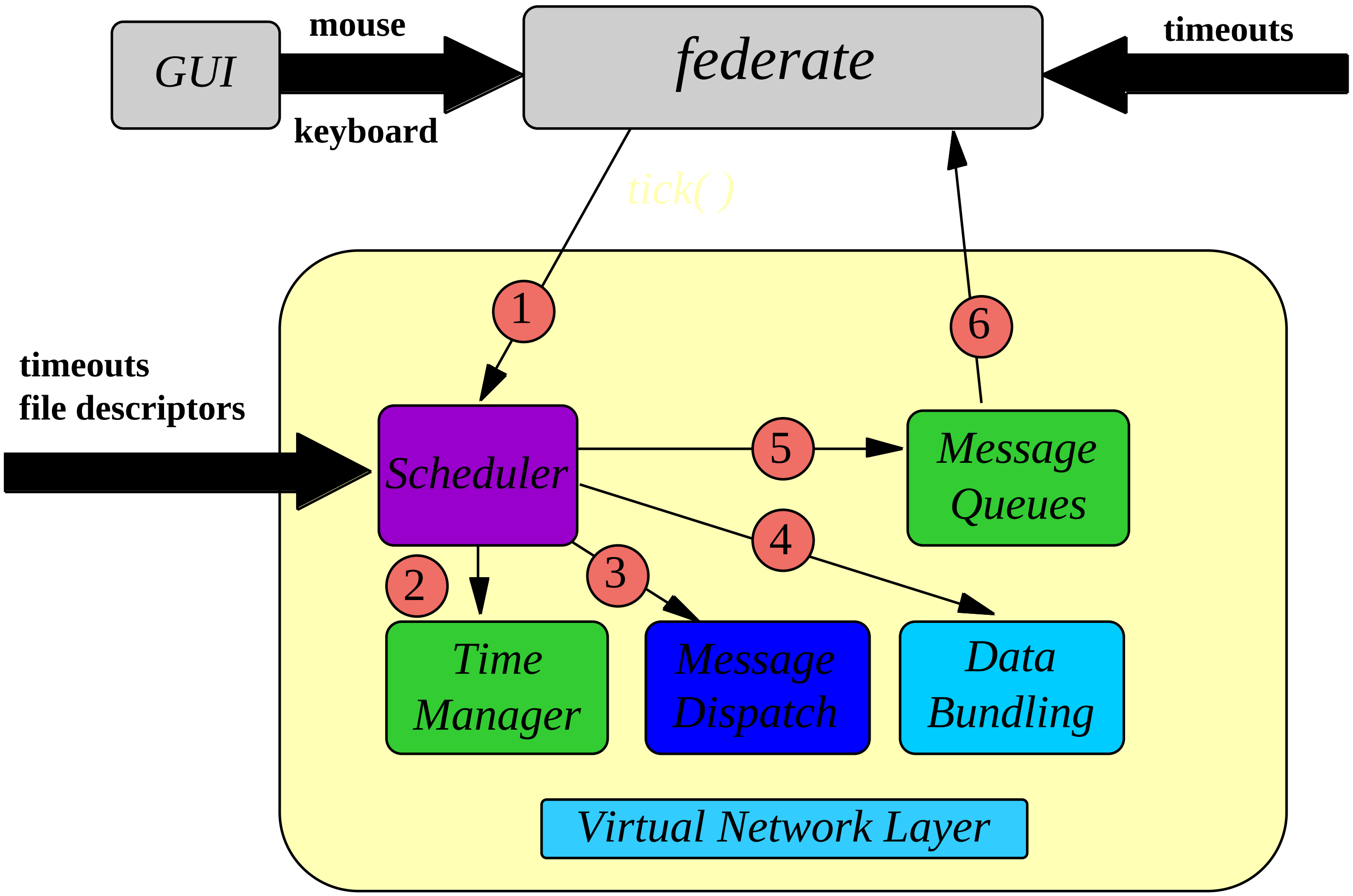
Sources of Asynchronous Events

- **Event Types**
  - **(1) File descriptor events**
    - **Input**
    - **Output**
    - **Exceptions**
  - **(2) Time events (Timeouts)**
- **Event Handling**
  - **<u>Polling</u> - inefficient (sometimes in the extreme)**
  - **<u>Event Unification</u>**
    - **Synchronous - “select( )” on multiple sources and block**
    - **Asynchronous - win32, Solaris 2.6, etc.**
  - **<u>Multiple Threads</u> - individual threads block on each event source**

RTI and Federate Event Sources



Single Threaded tick( )



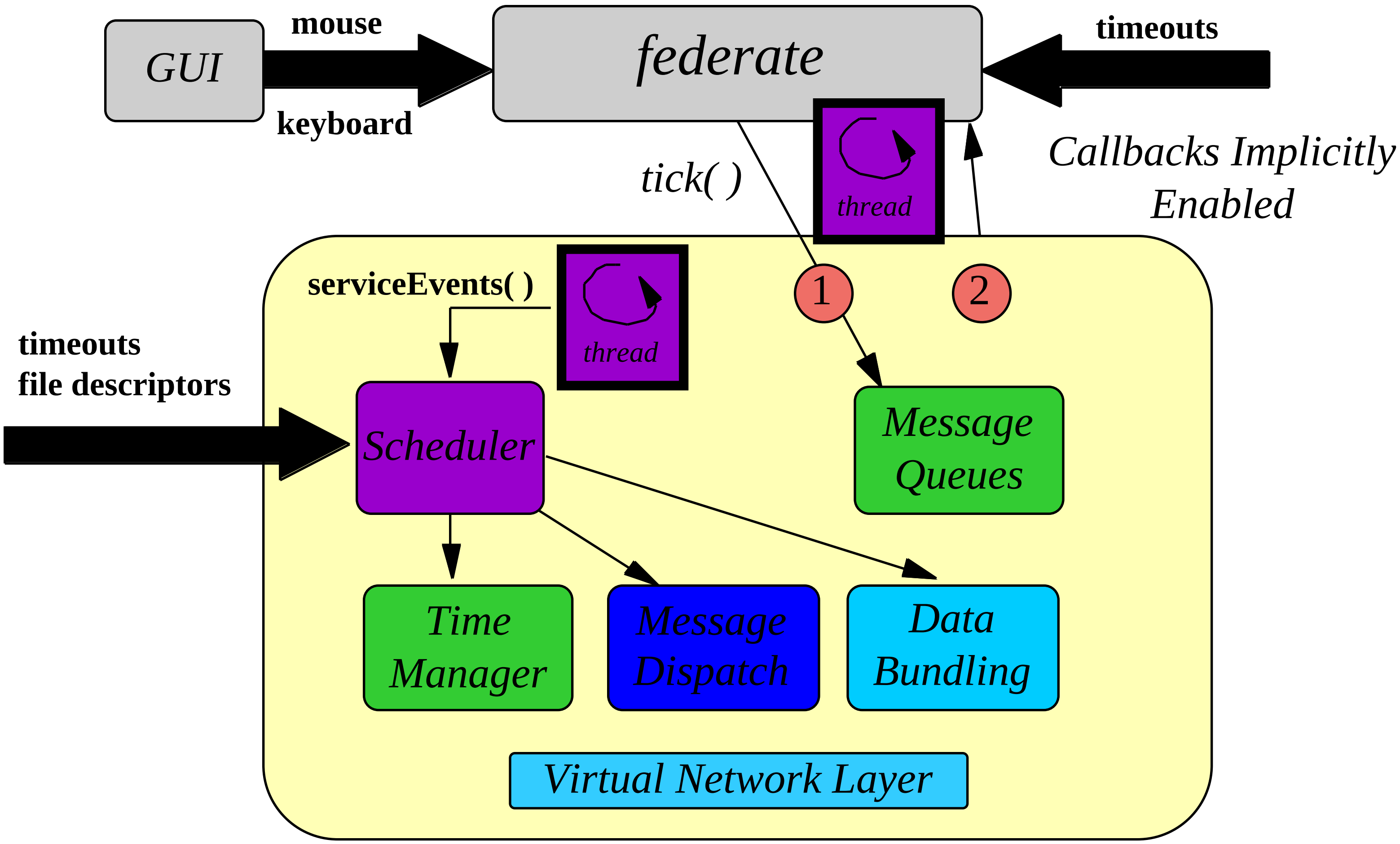
Issues Associated with tick( )

- tick( ) serves two purposes:
  - gives the RTI cycles to “service the wire”
  - gives the RTI cycles to “call back” the application
- Servicing the wire requires high frequency polling
  - current model means high-frequency state updates
  - difficult to sprinkle calls to tick() throughout Federate code
    - short duration computation loops OK
    - long duration computation loops may need to tick() throughout
      - must anticipate state updates (often difficult)
- disabling callbacks helps
  - federate frequently calls tick() with callbacks disabled
  - enables callbacks when able to service them

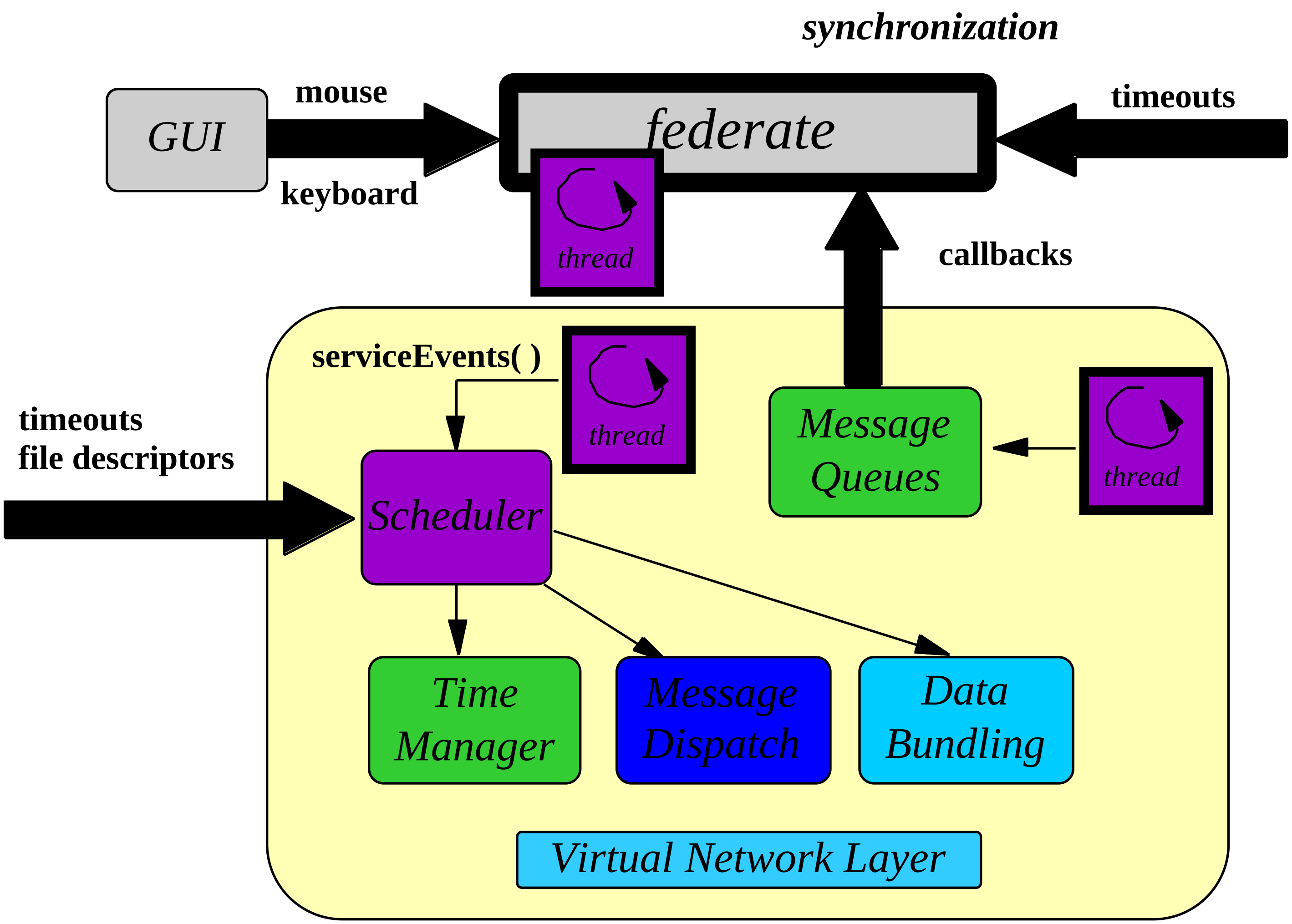
Threading Paradigm is a Match for Asynchronous Event Handling

- **Operations performed by by tick() with callbacks disabled are essentially decoupled from the Federate**
  - **We could decouple the processing with multiple threads**
- **Threading can improve performance depending on application characteristics**
  - **Advantages**
    - **Simpler to code Federate**
    - **OS schedules processing based on requirements**
  - **Disadvantages**
    - **expense due to context switching**
    - **expense due to synchronization locks**
  - **Single CPU (Preemption) - Federate need not call tick( ) based on worse case servicing requirements**
  - **Multiple CPU = Parallelism**

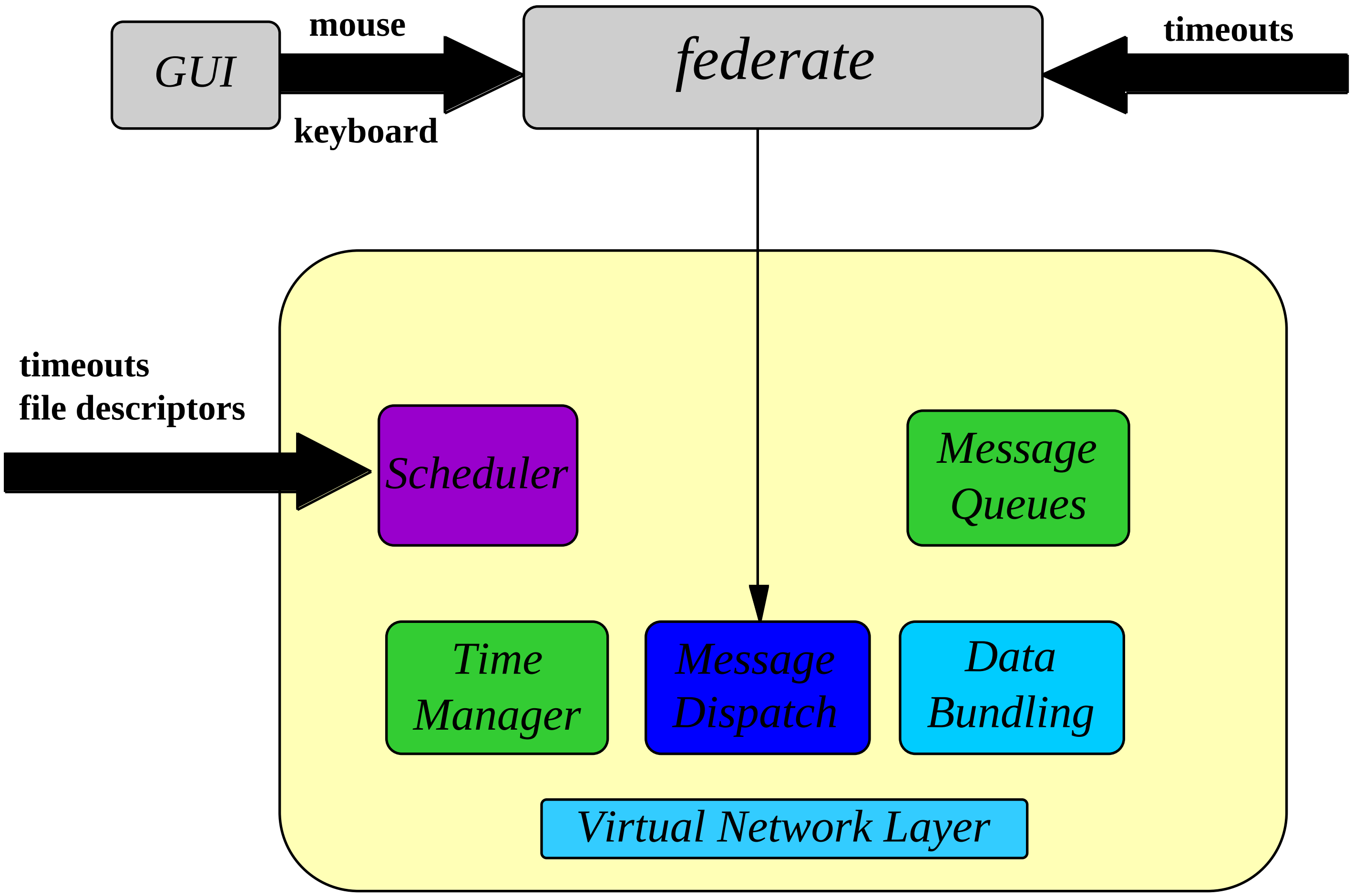
Thread in (non-Reentrant) Federate, Thread in RTI



Thread in Federate, Thread in RTI, Thread for Callbacks



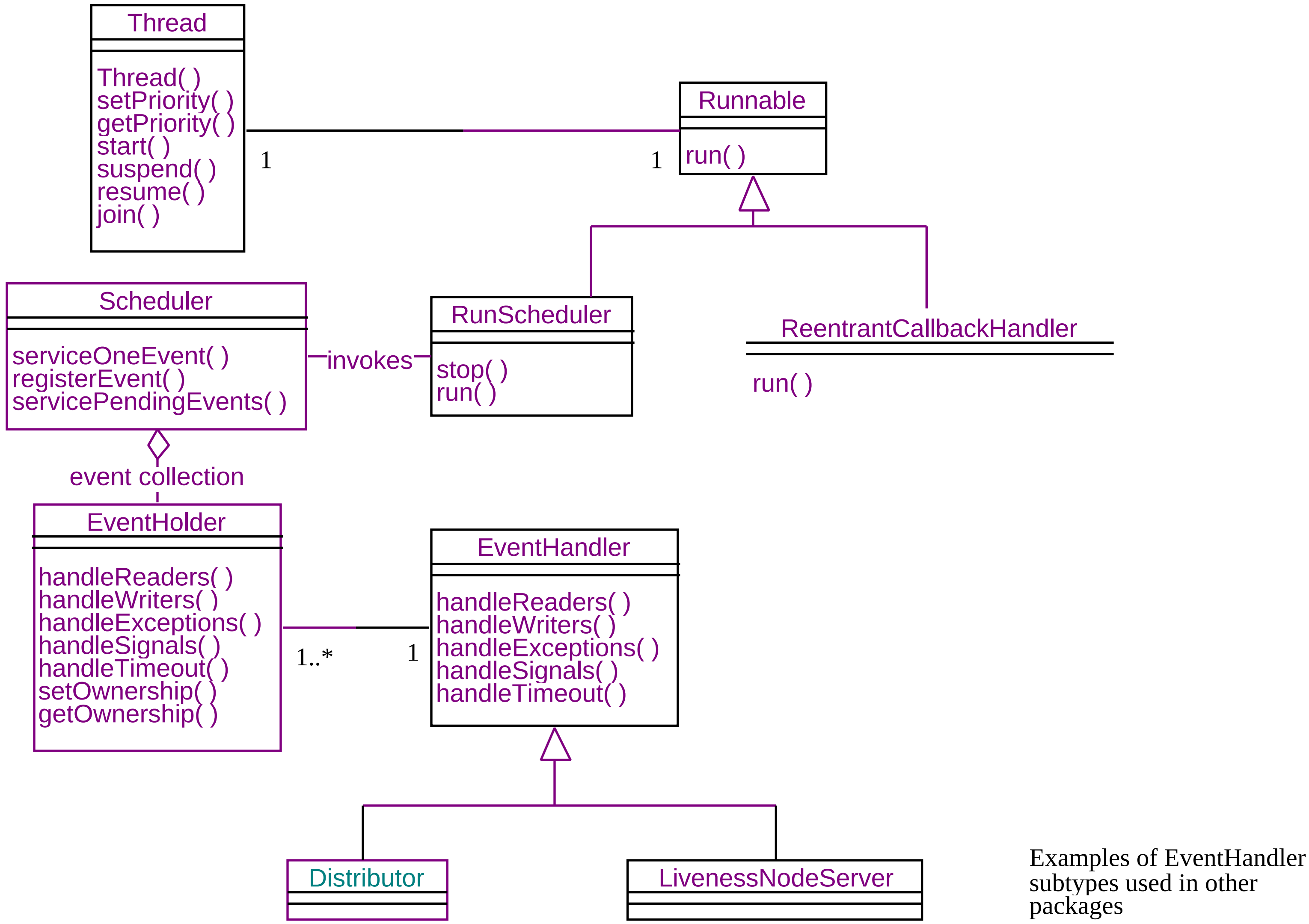
In All Cases, Calls to the RTI "Borrow" the Federate Thread (no context switch)



Configurable Threading Model

- **The RTI is designed to be tunable/configurable to the underlying thread model which the application developer desires**
  - **Allow threads where desired for maximum flexibility and throughput**
  - **"*Configure it out*" when not desired to eliminate synchronization/ context switching overhead**
- **Polymorphism and the Scheduler are the key**
  - **Implement components behind well defined "interfaces"**
  - **Individual threads respond to events using the Reactor Pattern**

Process Model Key Classes

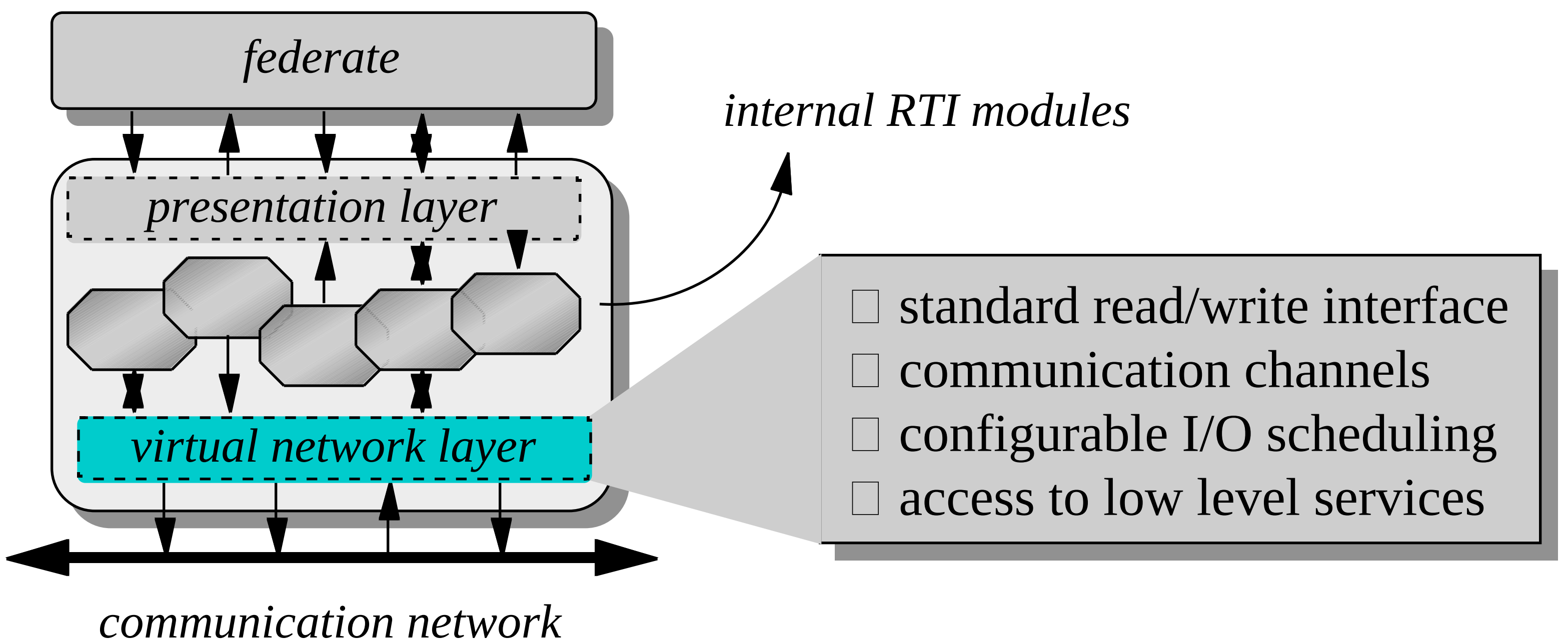


RTI 2.0 Design Modules

- Presentation Manager
- Administration
- Process Model
- Virtual Network
- Object Management
- Distribution Management
- Time Management

Virtual Network Module

- **Communication Module Requirements**
  - **isolate internal RTI modules from platform dependencies associated with I/O mechanisms**
  - **define standard interface to accommodate different transport protocols while exploiting platform advantages**



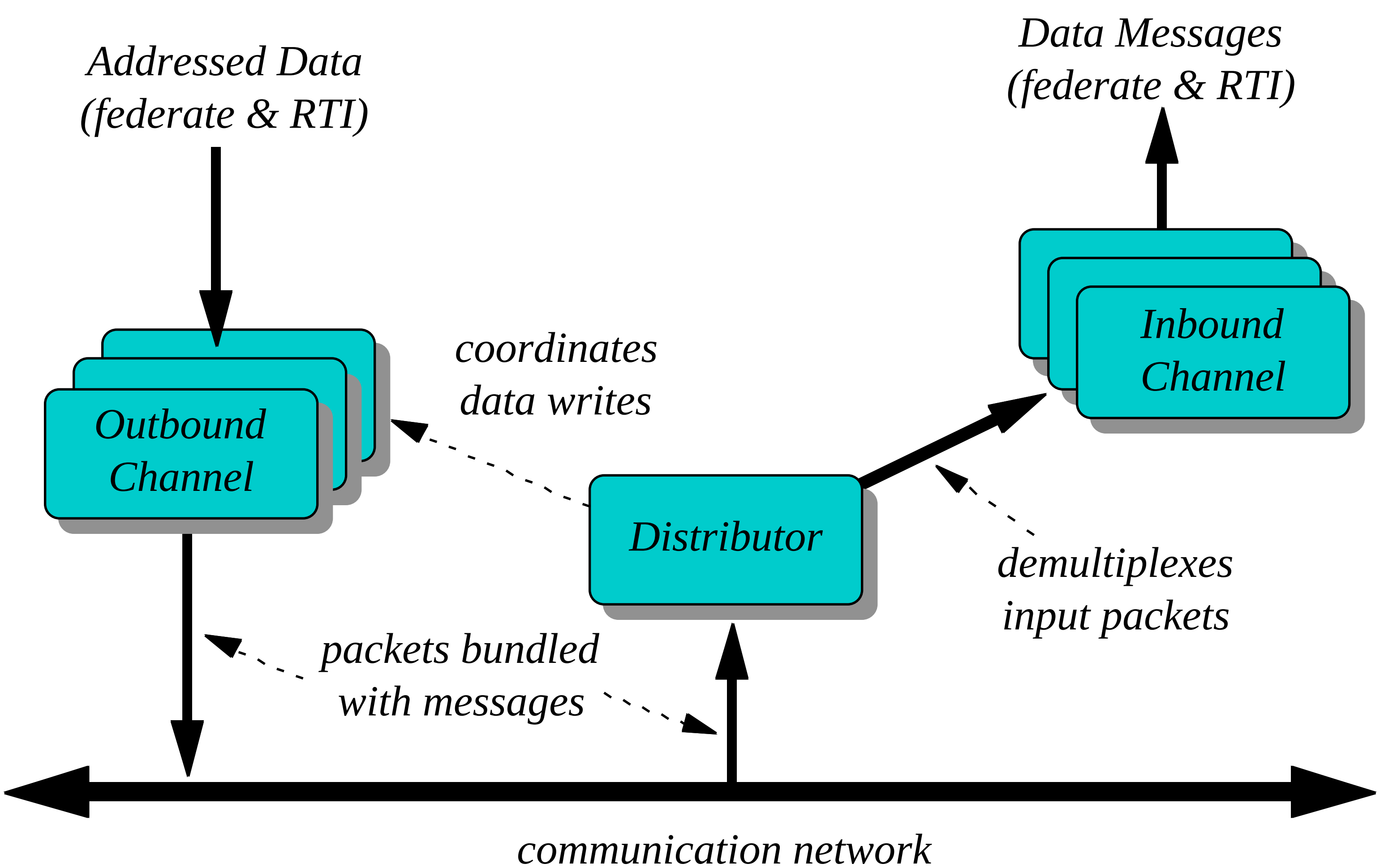
RTI Transport Requirements

- **Federation and RTI Traffic**
  - **federation traffic point-to-multipoint**
  - **RTI traffic point-to-point and point-to-multipoint**
  - **both types will have *best effort* and *reliable* transport requirements**
- **Best Effort**
  - **no acknowledgment of receipt necessary (UDP)**
  - **multicast technology offers best performance and scalability, but there are limitations to number of addresses supported**
- **Reliable**
  - **acknowledge data receipt (TCP, reliable multicast)**
  - **reliable transport may require different characteristics based on desired transport latency, acknowledgment latency, scalability**

Virtual Network Design Issues

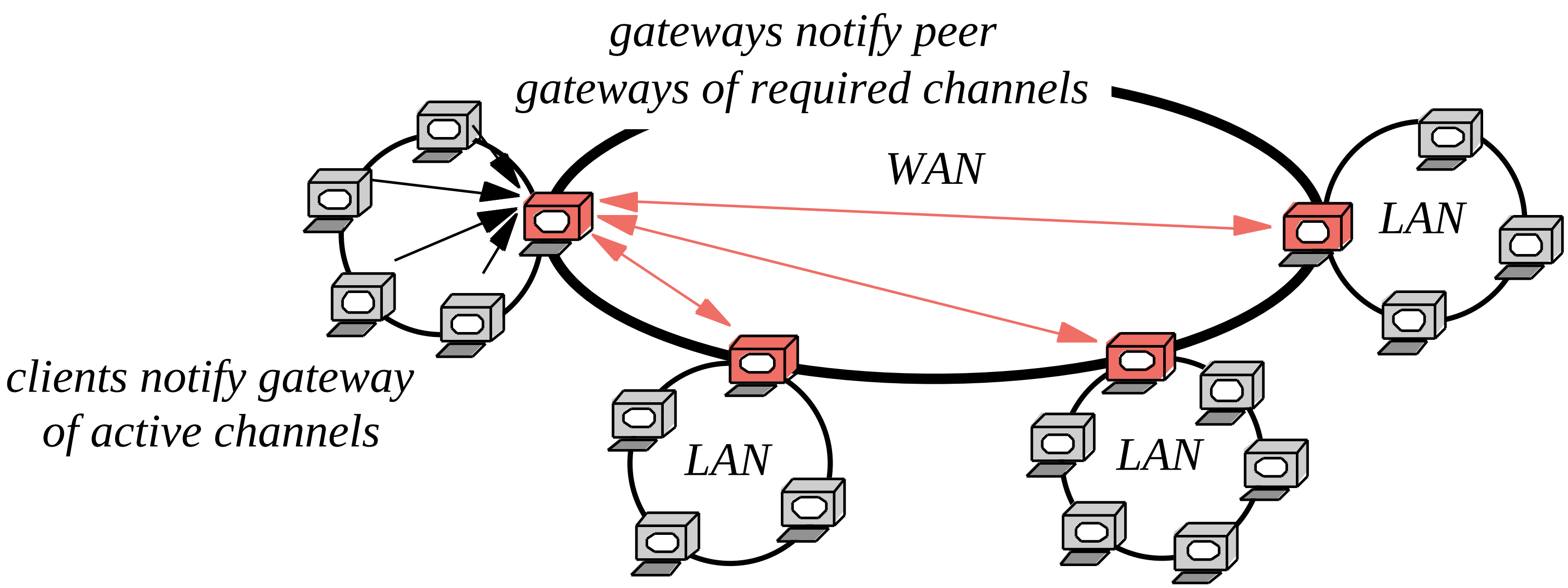
- **Performance**
  - getting data to consumers reliably is not difficult, its getting it there efficiently that must be accomplished
- **Portability**
  - diverse platforms and operating systems may have differences in low level I/O APIs, and provide non-standard mechanisms which can be exploited to improve performance
  - Flexibility
  - federations require configuration of communication channels
- **Extensibility**
  - future communication mechanisms and protocols need to be accepted in an efficient and maintainable fashion

Virtual Network Design Overview



Multilevel Distributors

- **Hierarchical Data Routing**
  - **hierarchy added for large-scale federations to increase scalability**
  - **software support currently required, but commercial hardware may support this functionality in the future**
  - **gateway unifies data interests to restrict inbound/outbound data, reduces multicast stress on WAN**



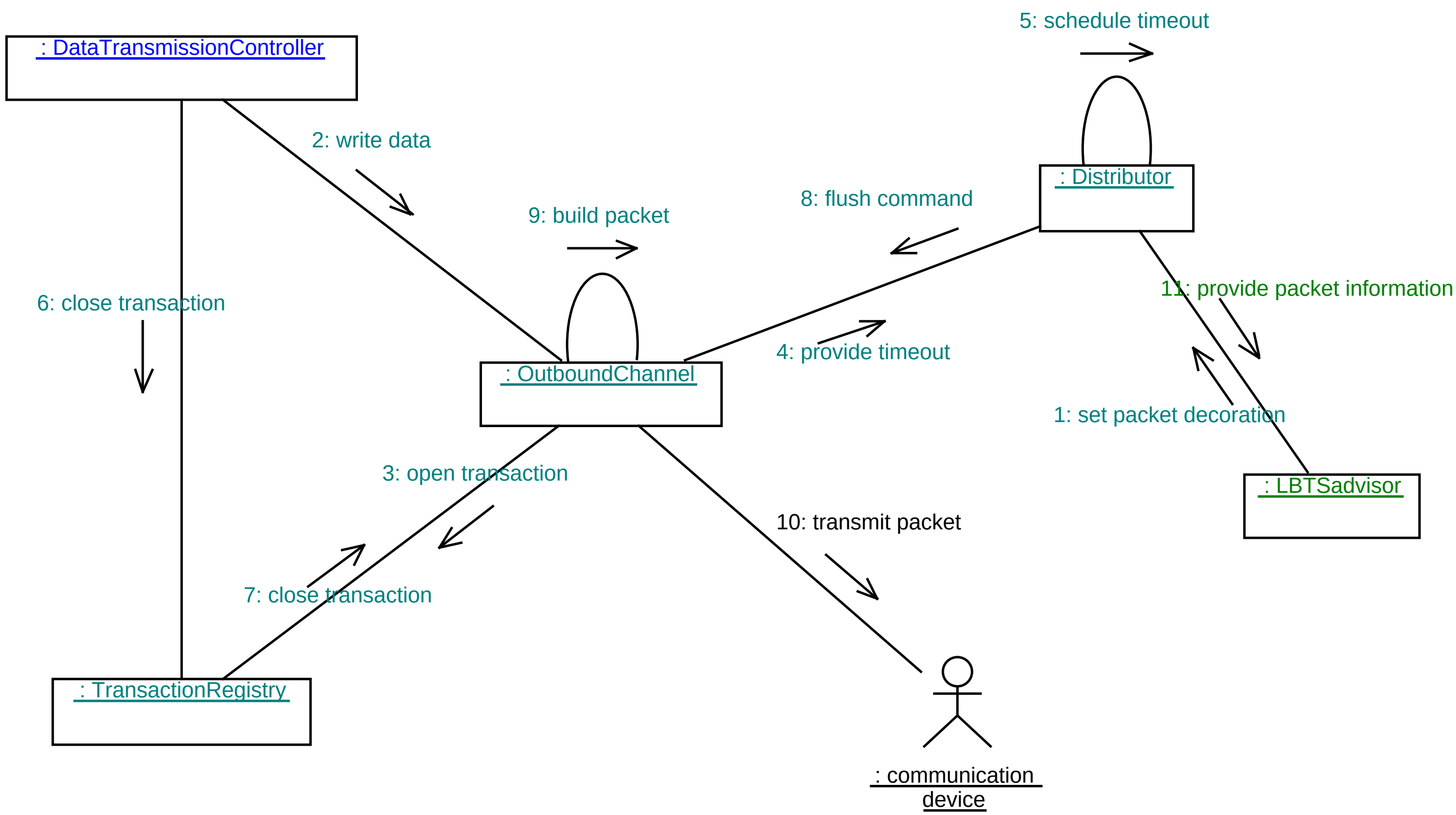
Key Virtual Network Classes

- **Channels**
  - ***physical*** **data stream abstraction, hides protocol details**
  - **more advanced or different mechanisms can be encapsulated**
- **Outbound Channels**
  - **simple write/flush interface**
  - **default bundling/fragmentation characteristics and priority (a particular message can override bundling times or priority)**
  - **message transaction control for individual attribute writes**
- **Inbound Channels**
  - **simple read interface**
  - **unpacks and reassembles into typed data messages**
  - **initiates processing of messages (polymorphic behavior)**

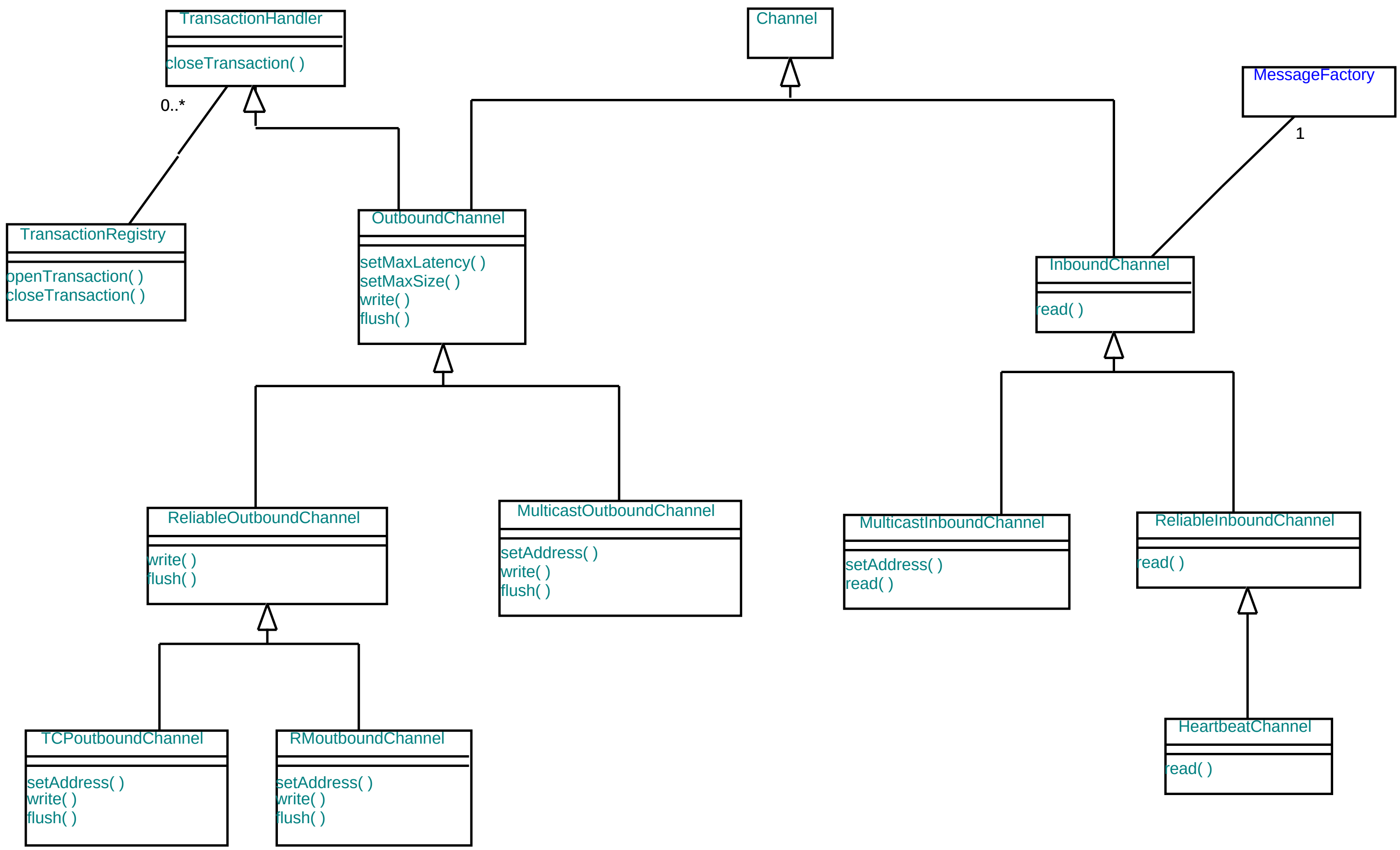
Key Virtual Network Classes (cont)

- Distributor
  - exploit OS and platform support (e.g., Solaris Asynchronous I/O, NT Completion Port)
  - coordinate data writes for the various outbound channels
    - priority aware, efficient time-out table mechanism
  - perform initial packet filtering on incoming data (e.g., federation handle, channel id)
  - demultiplex packets into inbound channels (priority aware)
- Channel Factory
  - responsible for creation of all channel objects
  - uses hierarchical type and marker strings

Data Transmission Collaboration Diagram



Outbound/Inbound Channel Class Diagram

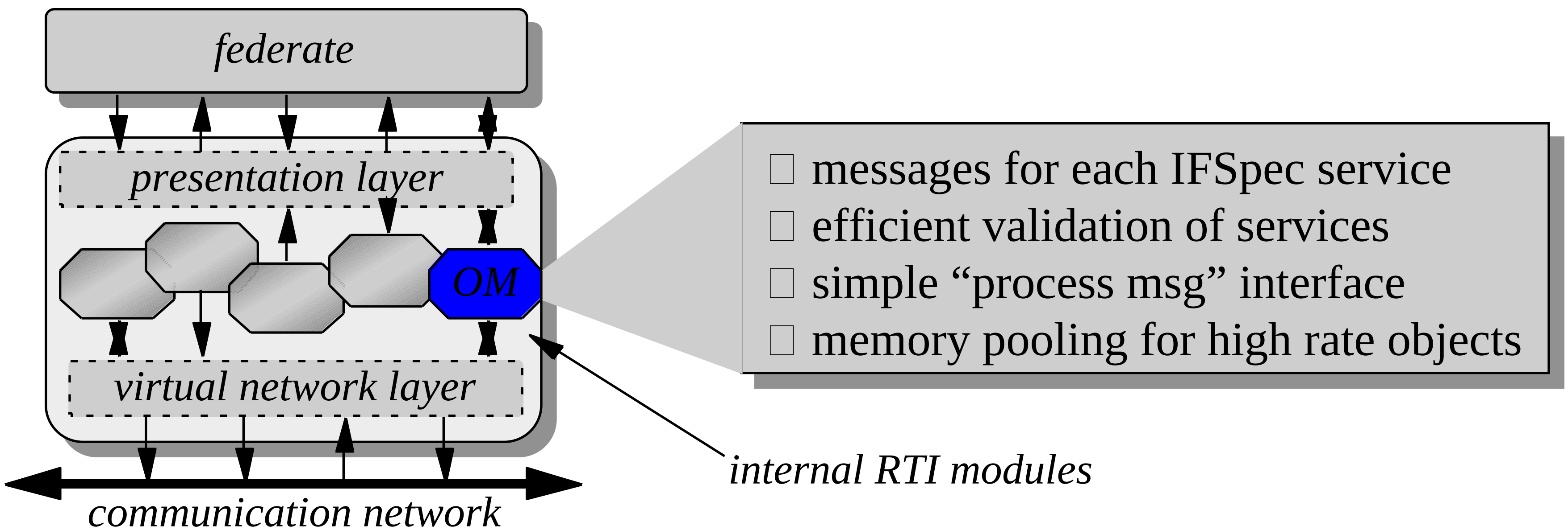


RTI 2.0 Design Modules

- Presentation Manager
- Administration
- Process Model
- Virtual Network
- Object Management
- Distribution Management
- Time Management

Object Management Module

- **Federation Objects & Interaction Requirements**
  - **maintain FED information and perform RTTI**
  - **implement protocol for IFSpec services**
  - **maintain database of known objects**
  - **maintain publication, subscription, and ownership information**



Support for Efficient Data Flow

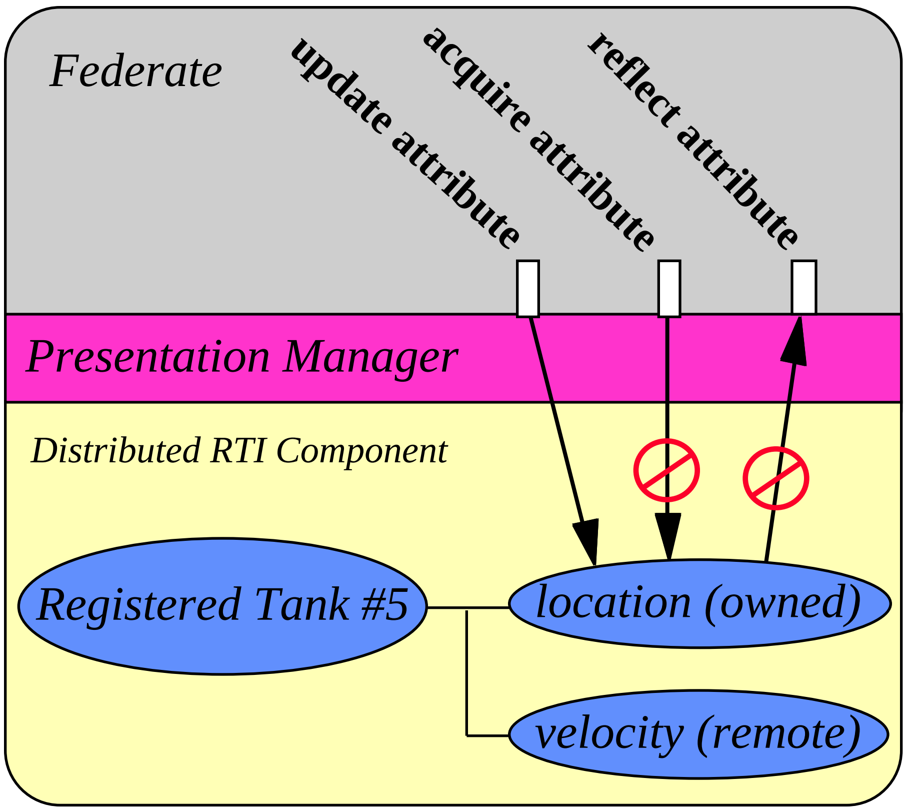
- In order to have a high performance RTI we need to optimize the primary data flows for attributes and interactions
- Performance factors affecting efficient data flow
  - validation of service invocations (checking pre-conditions)
  - data copies / memory management

Object Management Design Issues

- **Performance, Performance, Performance!!!**
  - **must optimize primary data flows for a high performance RTI**
    - **eliminates unnecessary copies and memory allocation/deallocation through message objects**
    - **eliminates unnecessary checks for services through DTC and state objects (while still catching invalid invocations)**
- **Extensibility**
  - **design must be extensible to easily adapt to the evolving HLA standard**
    - **extensibility is built into design through abstract interfaces, factory classes and factory methods**

Validation of Service Invocation

- **Issue(s)**
  - **state of attributes and interactions determines validity of services**



- federate can update tank #5's location
- federate can not request attribute ownership acquisition on tank #5's location
- federate can not reflect tank #5's location attribute

Validation of Service Invocation: Alternative #1

- **Enumerate states and use switch statement for each service invocation**
  - **use of conditional statement to implement state specific behavior**
    - **compiler optimized check O(1)**
  - **replicated conditional statements exist through out the system**
    - **decreases maintainability of system**

```
function updateAttributeValues(...)
{
  get attribute
  switch (attribute.state)
  {
        case PublishedOwned :
                sendToDestinations;
        case SubscribedNotOwned :
                throw exception;
  }
}
```

Validation of Service Invocation : Alternative #2

- **encapsulation of state in an object that implements behavior for each service invocation**
  - **use of class to encapsulate state specific behavior**
  - **isolates all behavior for a state in a single class**
    - **increases maintainability of system**

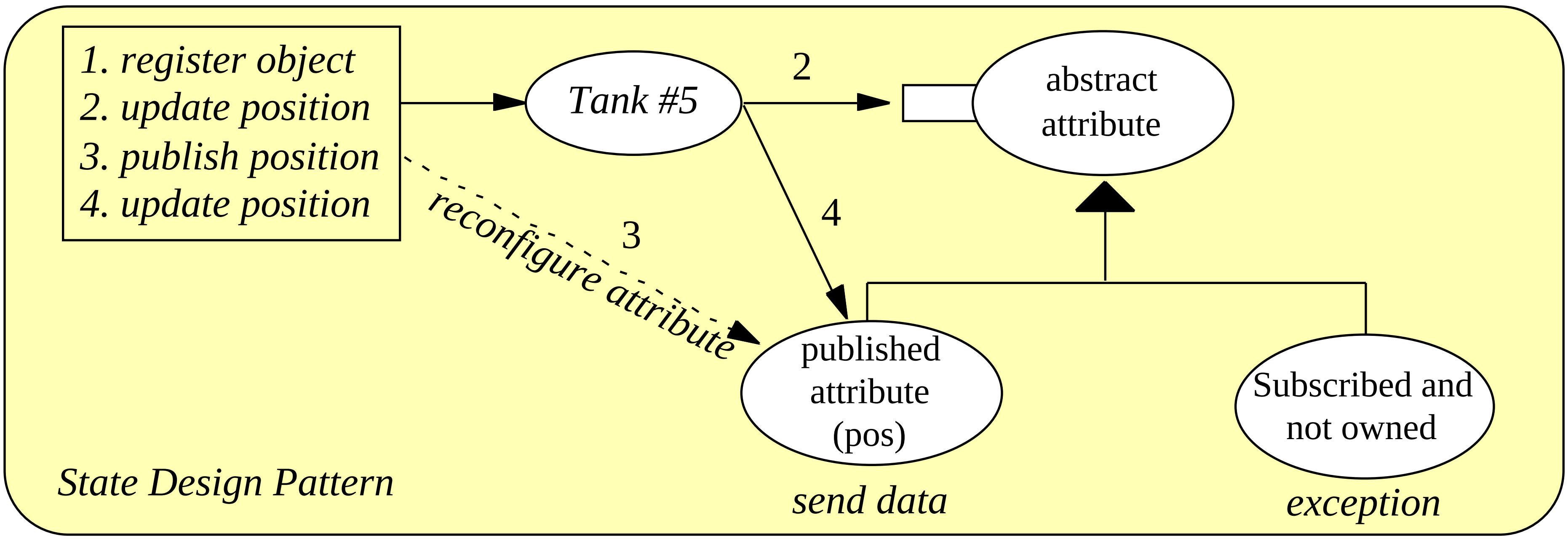
```
class SubscribedNotOwned
  : AttributeState
{
  processUpdateAttributeValues()
  { throw exception };

  processReflectAttributeValues()
  { give to federateAmb };

  processRequestAttrOwnDivest()
  { throw exception };
}
```

Validation of Service Invocation : Architectural Choice

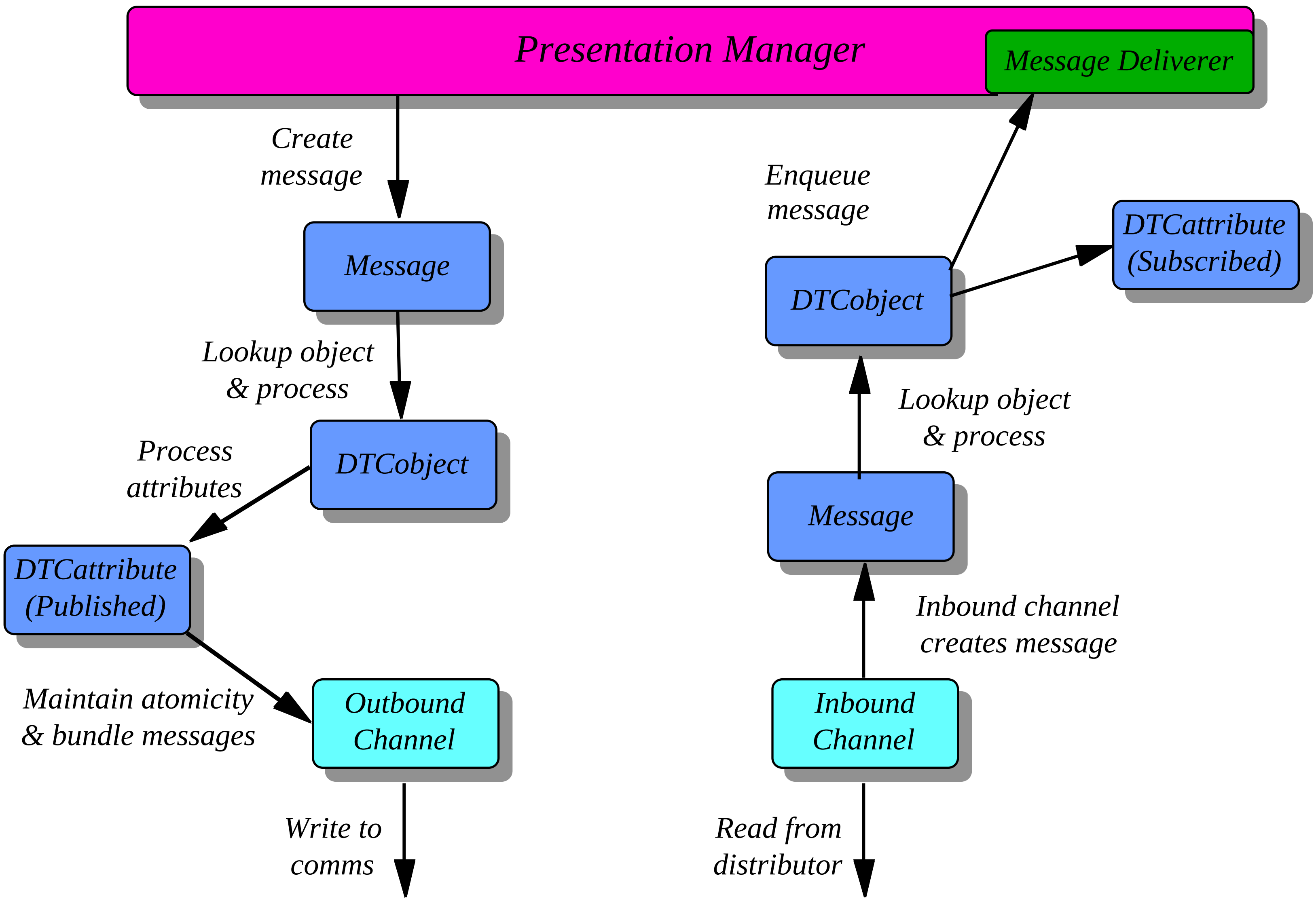
- **Alternative #2 was chosen in the design:**
  - **it better encapsulates the behavior for a given state**
  - **it eliminates the need to replicate a switch statement in each of the functions that implement an HLA service (errors often arise from missing cases in the switch statement)**



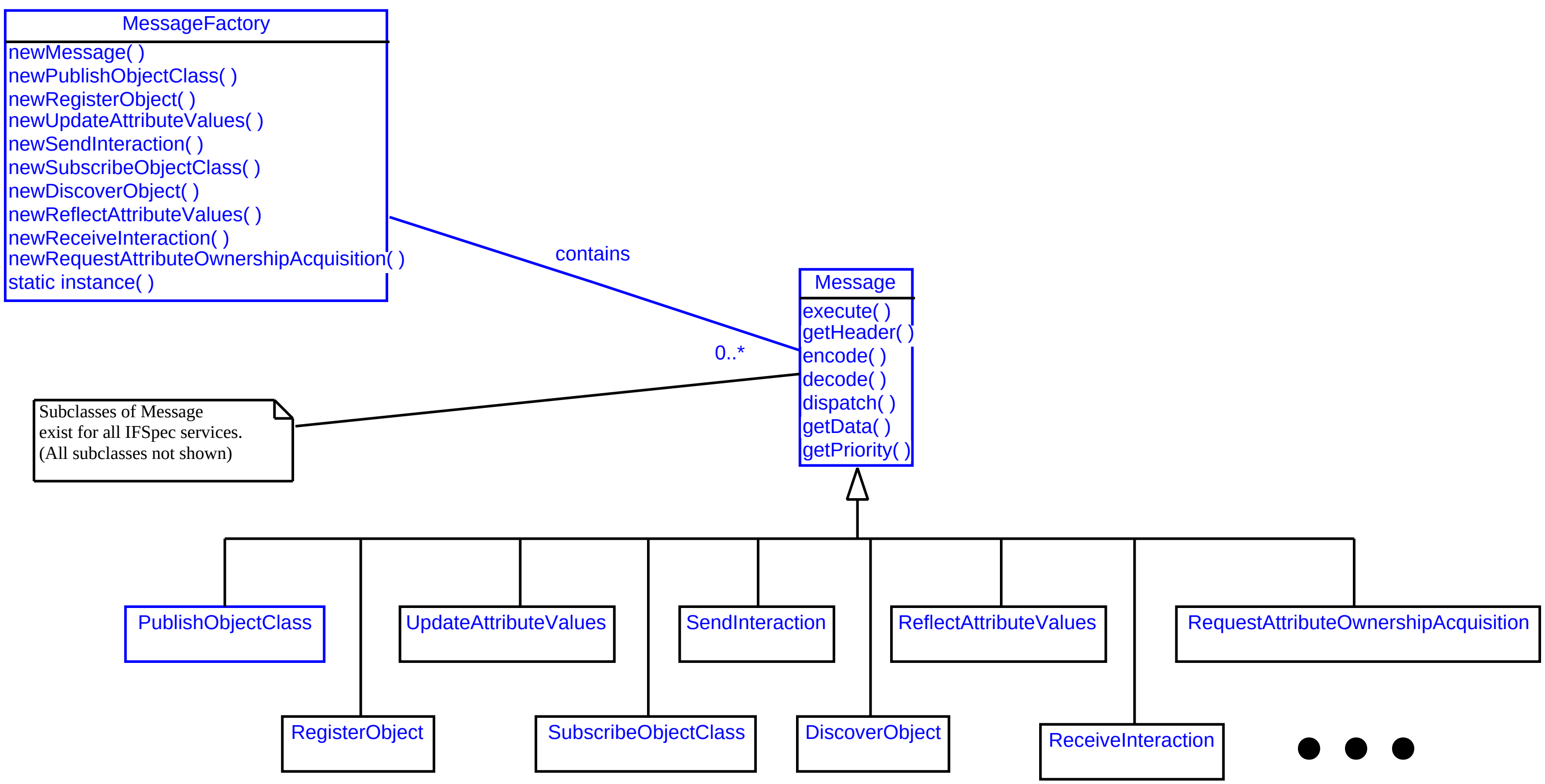
Data Copies / Memory Management

- **Data copies**
  - based on analysis and optimization of service validation, federate thread goes "to the wire" for federate initiated calls
    - few CPU cycles are expended during federate initiated calls
    - "to the wire" means written to network or bundled in a channel
    - no copies are needed since data has been processed before function return
- **Memory Management**
  - "flyweight" pattern is used for large quantity objects
    - one instance of each type that does not maintain state
    - other objects use flyweight providing arguments to act on
  - object pools for high rate objects
    - objects are reused instead of allocated & deallocated over time

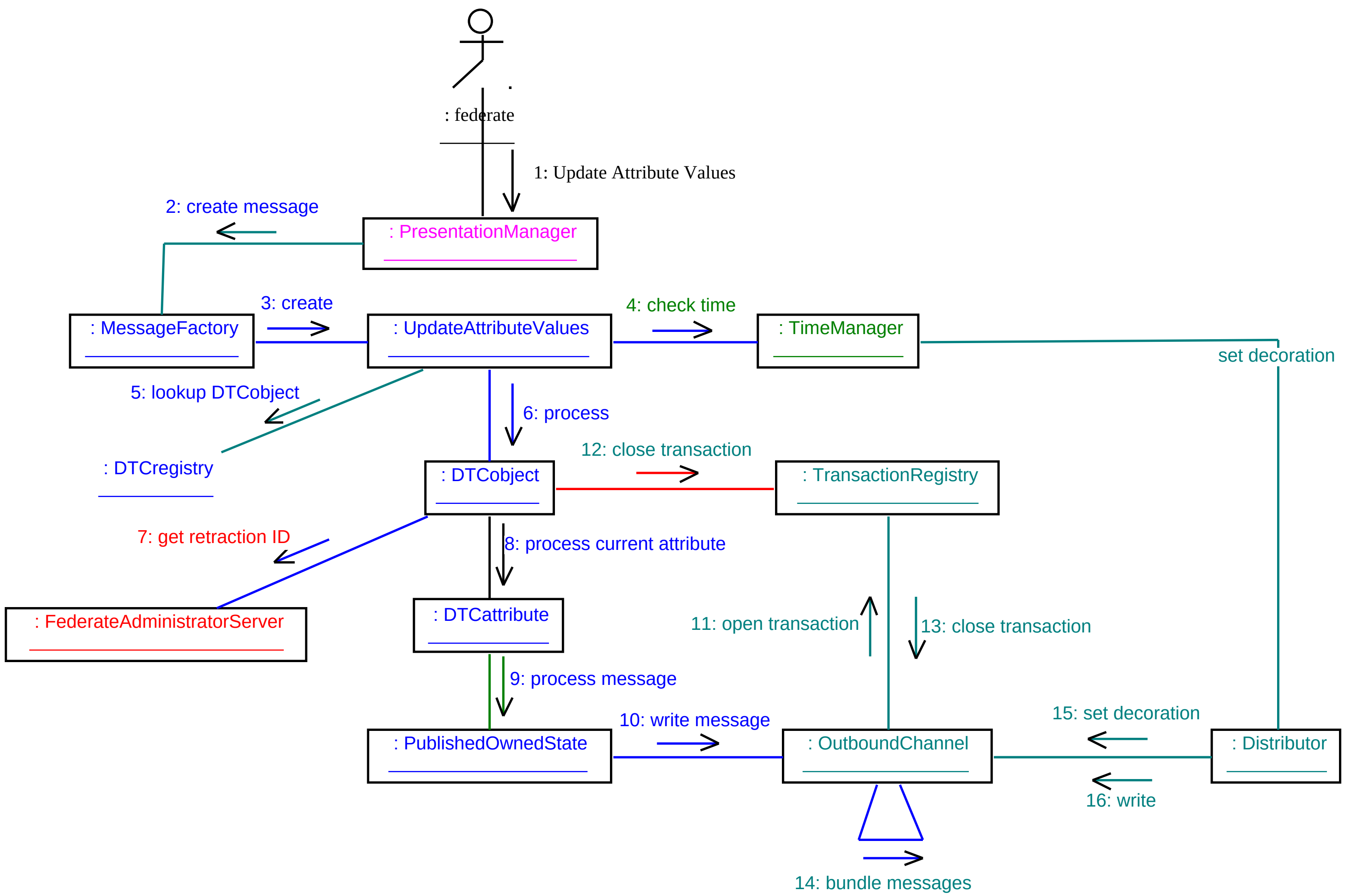
Object Management Design Overview



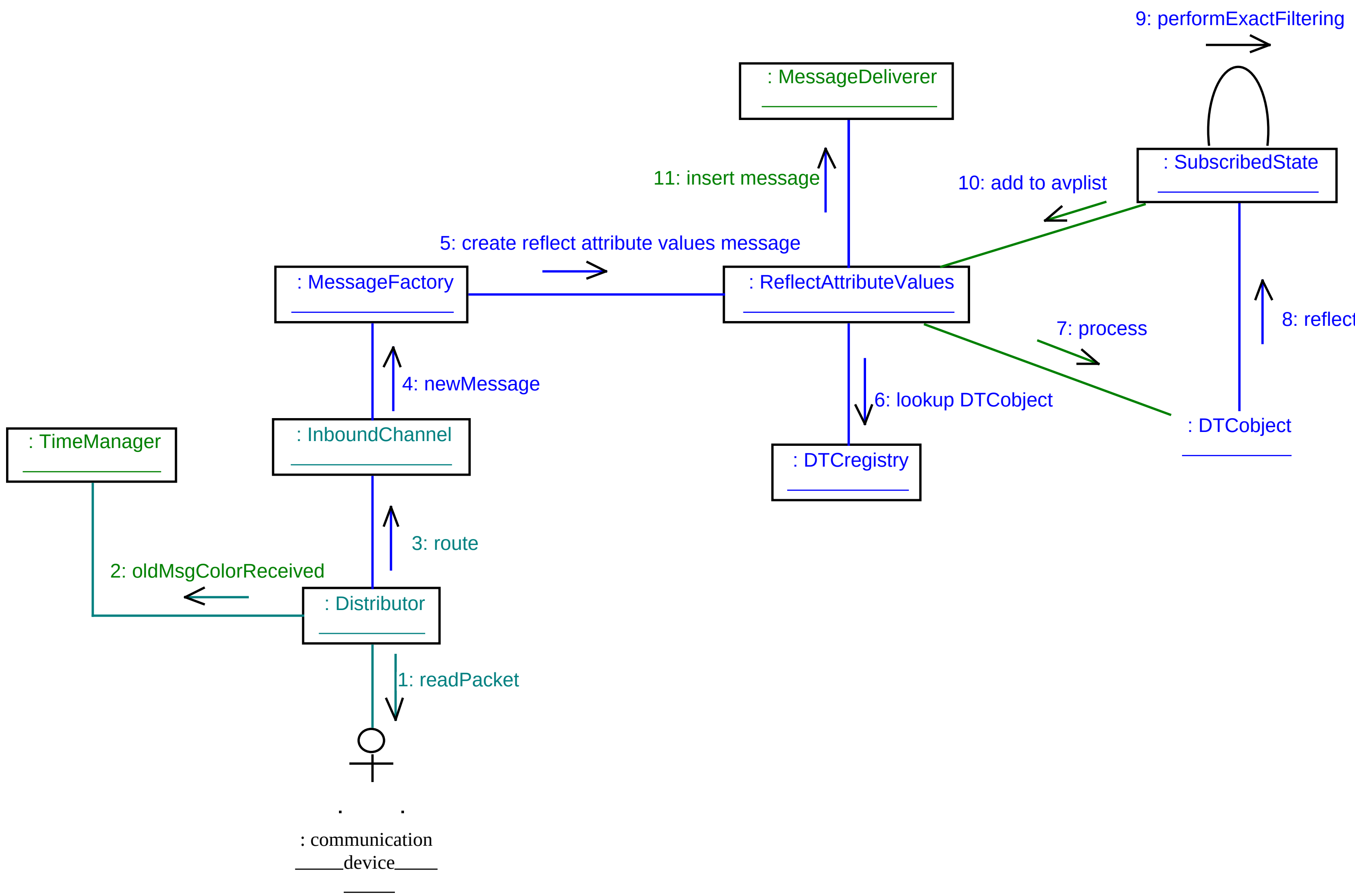
Message Protocol Class Diagram



Update Attribute Values



Reflect Attribute Values

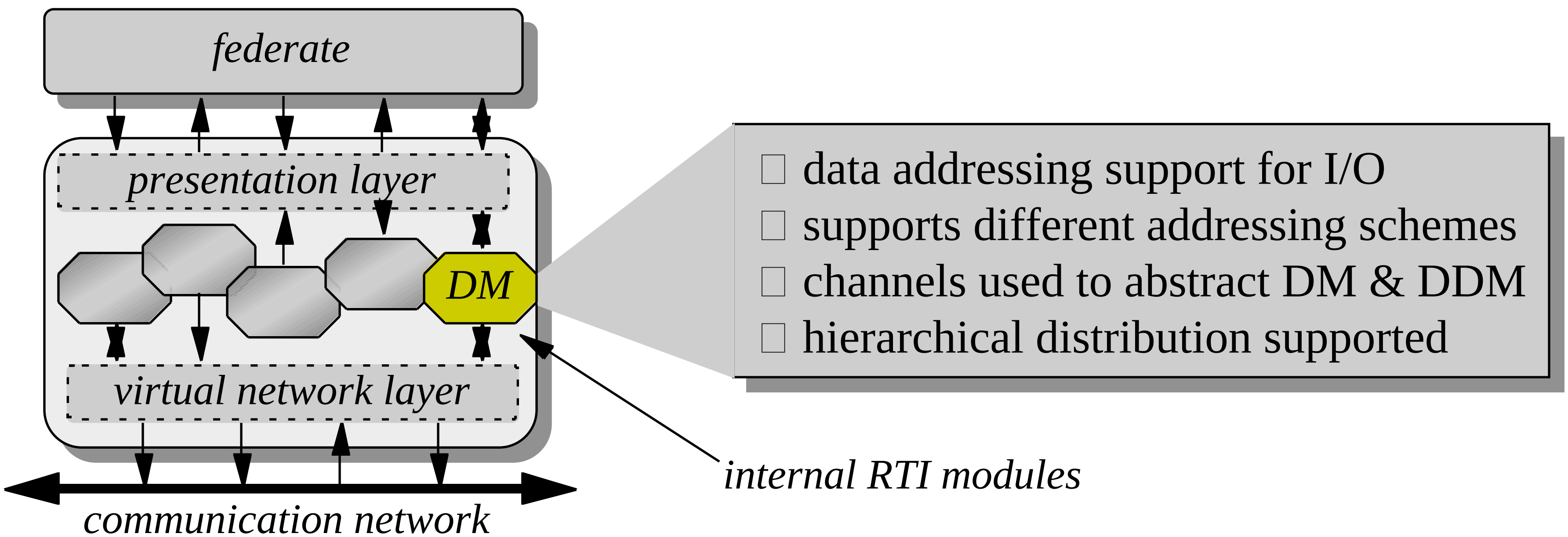


RTI 2.0 Design Modules

- Presentation Manager
- Administration
- Process Model
- Virtual Network
- Object Management
- Distribution Management
- Time Management

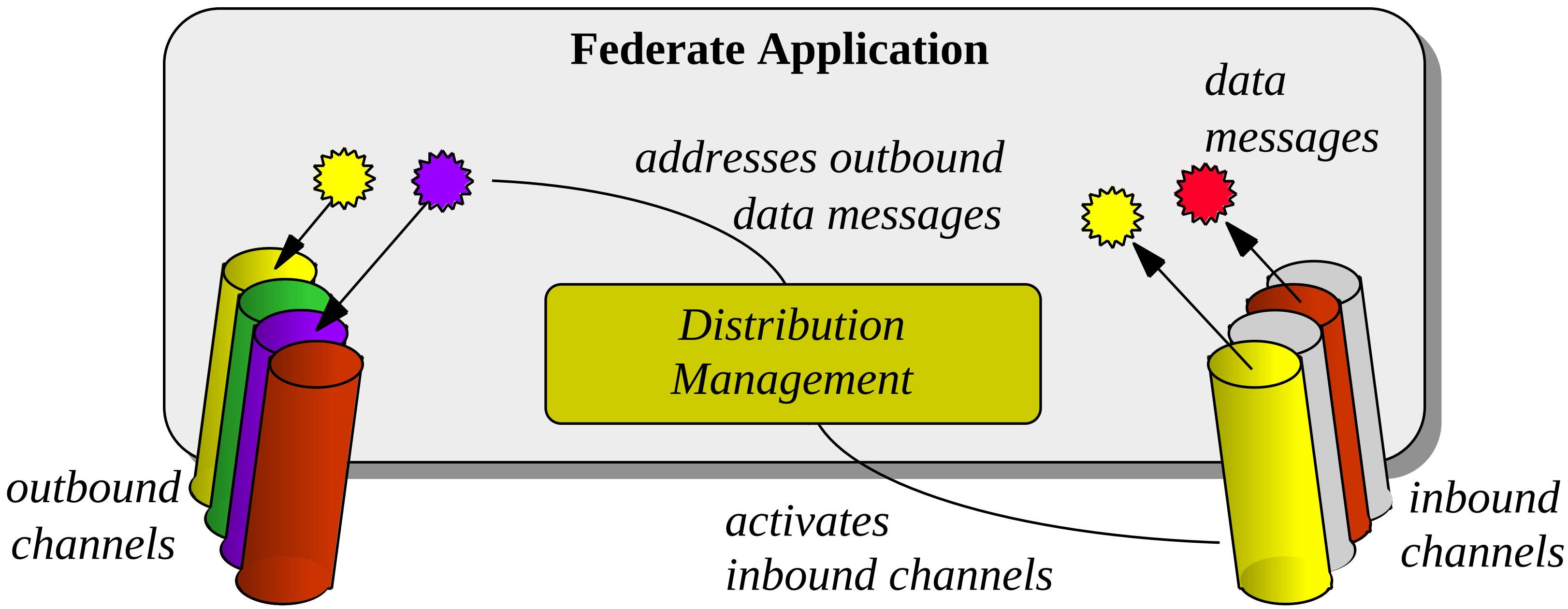
Distribution Management Module

- **Data Routing Requirements**
  - **support the use of class (DM) and value (DDM) based filtering to reduce the amount of unwanted traffic received by federates**
  - **configurable to the needs of the federation**



Distribution Management

- **Data Addressing & Routing**
  - **data needs to be routed along communication channels which are used to provide segmentation in order to reduce unwanted traffic**
  - **federations need the ability to customize the data routing scheme, which then manages channel creation and addresses data**



Data Addressing

- **RTI supports two types of filtering**
  - ***class based filtering*** **routes data solely on the type of attribute or interaction using the Declaration Management Services**
  - ***value based filtering*** **routes data according to rules that are dependent on particular data values (typically the values of the data being routed but not necessarily) using the Data Distribution Management Services**
- **Ideally an infinite number of addresses or channels would be used to provide perfect segmentation of data**
  - **unfortunately there are limitations on the number of communication streams that can be supported, and the computational resources required for addressing become extreme**

Value Based Filtering (Data Distribution Management)

- **Complete solution of dynamic producer-consumer matching requires continual access to global knowledge**
  - **and still results in an NP complete problem ☹**
- **Design Approach**
  - **initially implement *<u>general purpose static gridding scheme</u>*, but anticipate additional approaches architecturally**
  - **dynamic grid adjustments can be added based on:**
    - ■ **load-leveling: balance traffic loads across channels**
    - ■ **packet rejection: optimize amount of traffic mis-routed**
  - **“research-required” approaches are supported**
    - ■ **source-based addressing**
    - ■ **clustering**

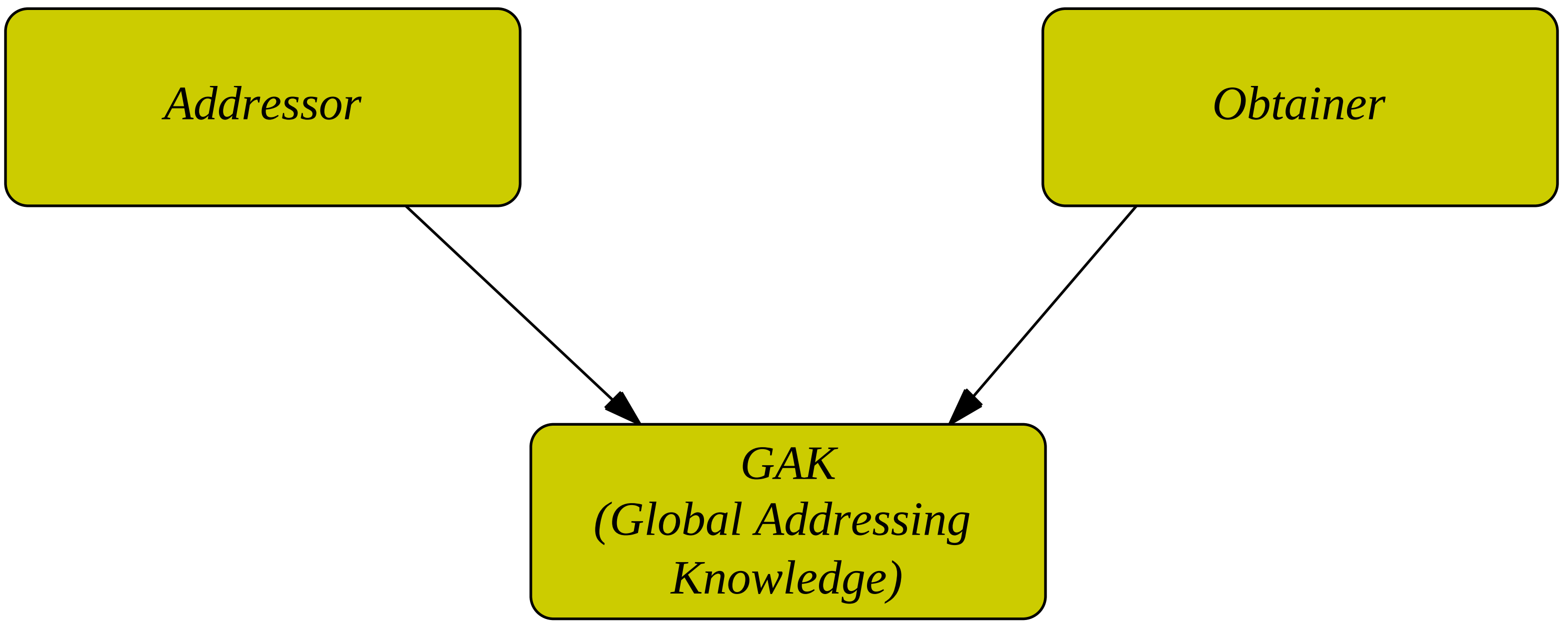
DM and DDM Issues

- DM and DDM Relationship
  - architecture designed to support DM/DDM unification if required (although this is not recommended by the design team)
- Perfect Filtering of Routing Space Regions
  - 1.1 spec implies that an overlap of update region and subscription region is required for data to be delivered to the federate
  - design is configurable to allow federation to choose
- Multiple Receptions
  - class and value based addressing may result in multiple transmissions, design will remove duplicate messages
- Resource Allocation
  - DM and DDM require key system resources, such as mcast groups
  - resources allocated across DM/DDM according to federation needs

Distribution Management Design Overview

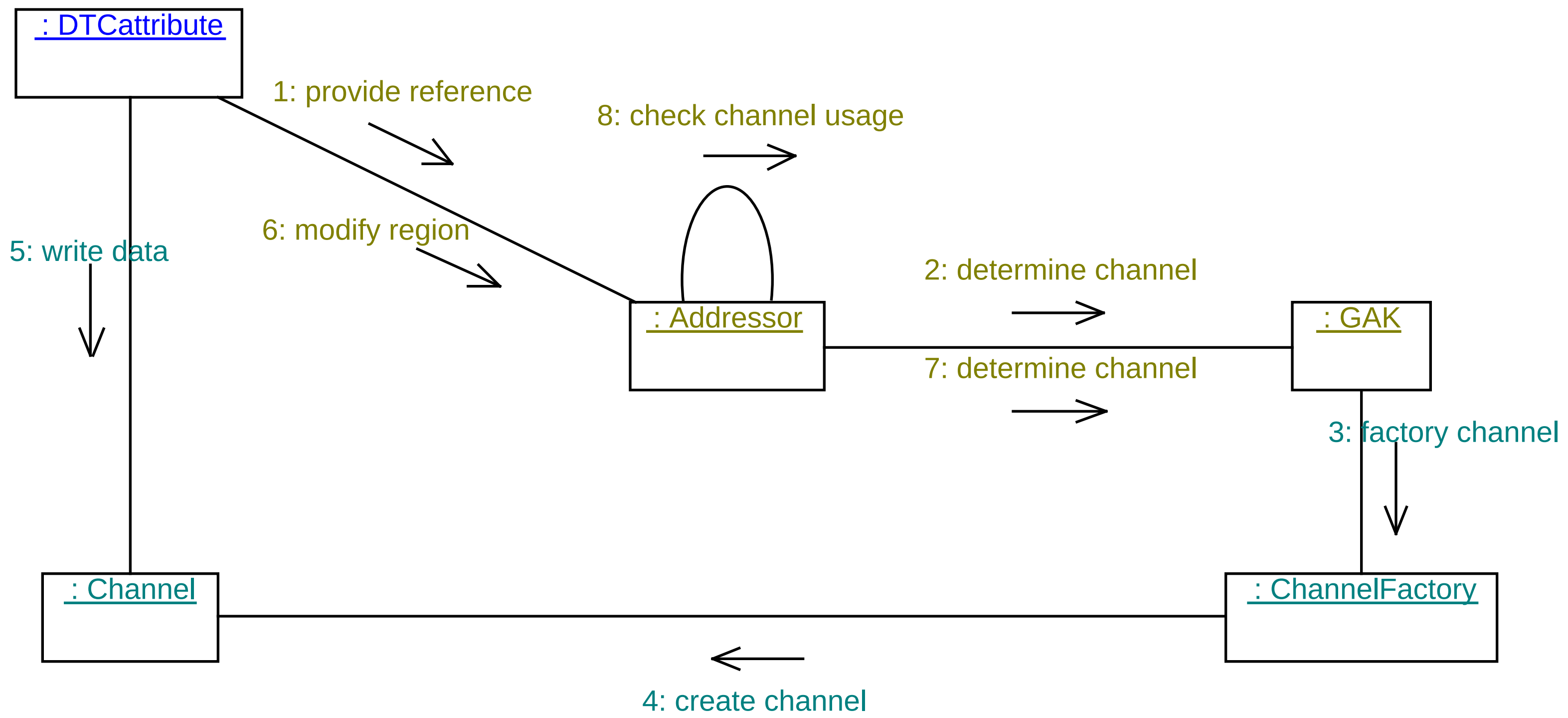
determines channel id
for attributes and interactions

activates and deactivates channels
based on subscriptions

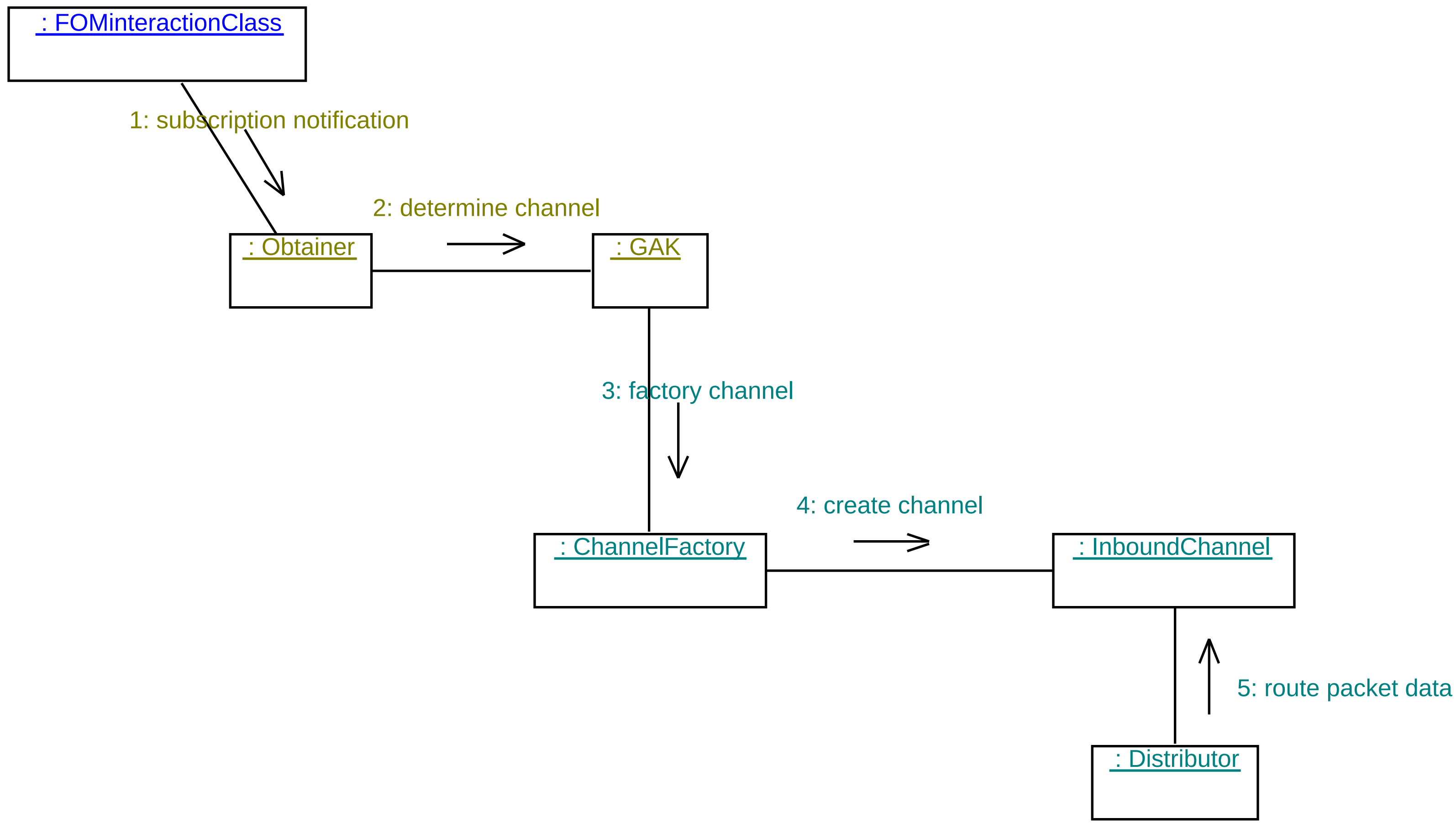


implements class and value
based addressing scheme

Data Addressing Collaboration Diagram



Data Subscription Collaboration Diagram

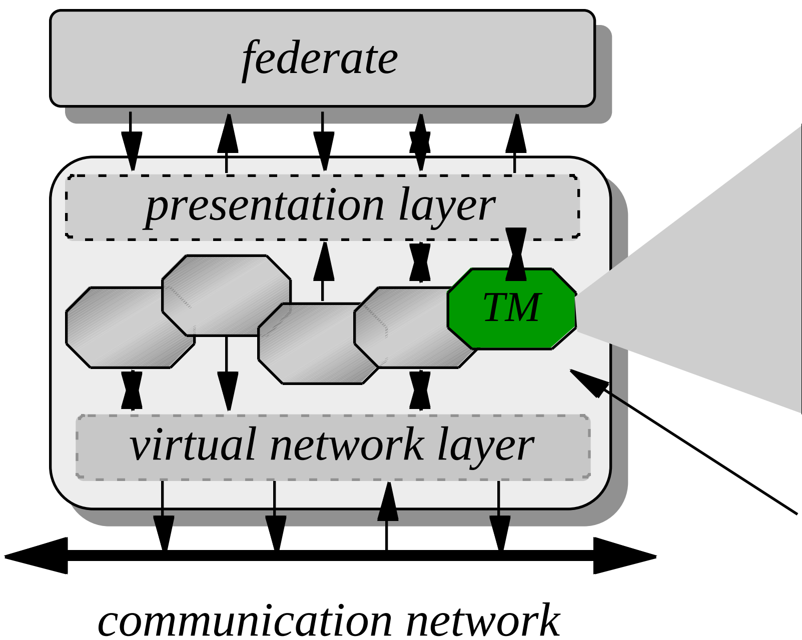


RTI 2.0 Design Modules

- Presentation Manager
- Administration
- Process Model
- Virtual Network
- Object Management
- Distribution Management
- ° Time Management

Time Management

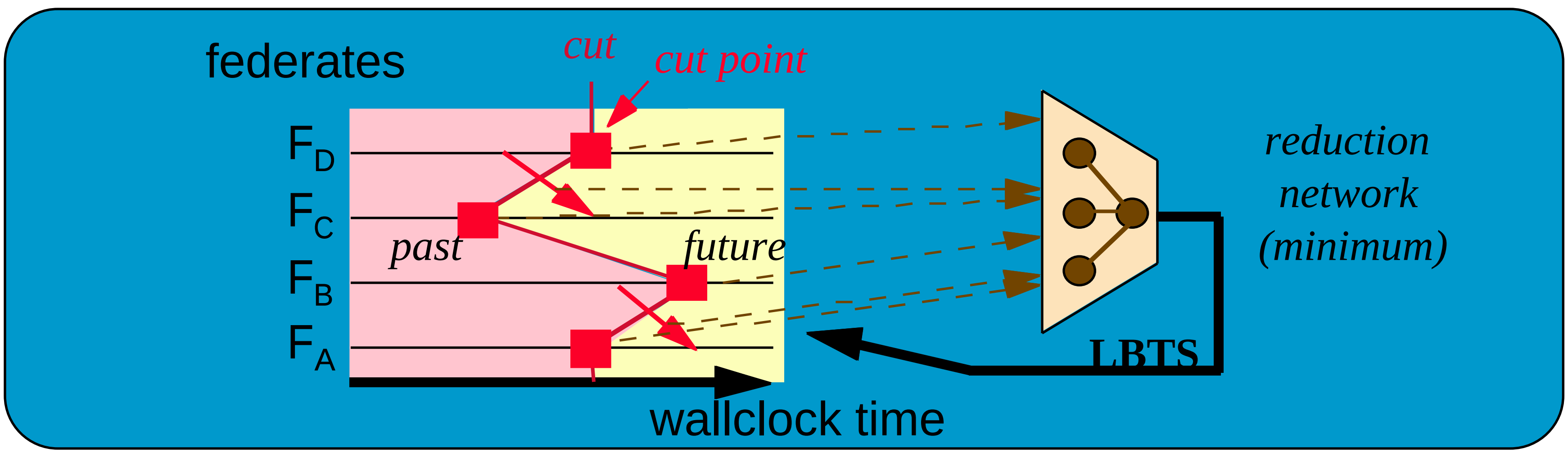
- **Requirements**
  - **implements the HLA time management services**
  - **accommodates anticipated specification changes**
  - **controls the advancement of a federates time**
  - **properly orders all information released to the federate**



- provides efficient message delivery
- asynchronous LBTS algorithm
- fault tolerance
- scalable for large federations

LBTS Algorithm Overview

- **LBTS algorithm's salient features**
  - **adaptation of Mattern's distributed snapshot algorithm**
  - **scalable to large federations**
  - **asynchronous to avoid artificial deadlock**
  - **handles zero-lookahead**
  - **fault tolerance support (timers and message color)**



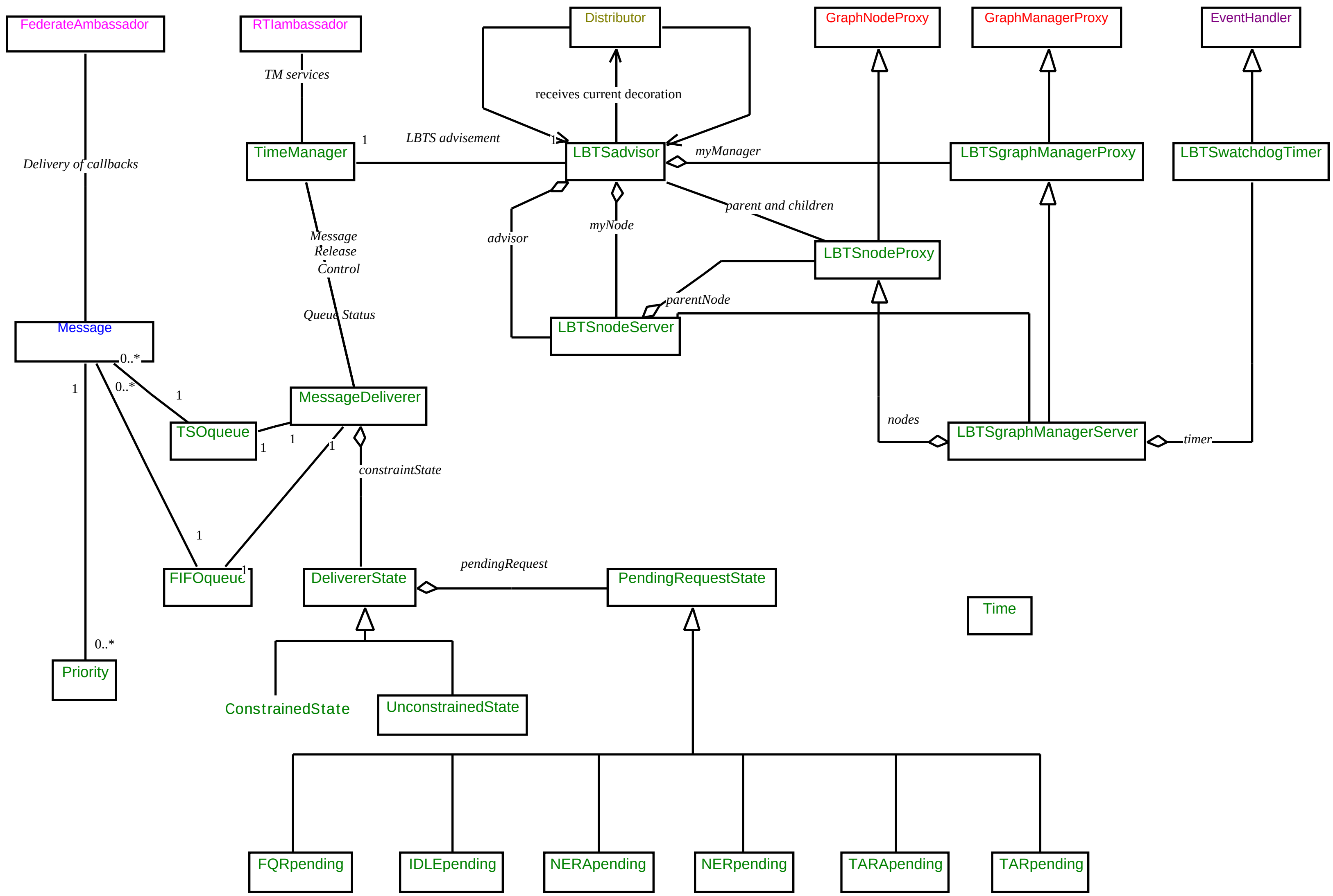
LBTS Algorithm
Communication Requirements

- **Reliable message delivery (Unordered delivery sufficient)**
- **Reduction tree support**
- **Number of destinations on outgoing messages required**
- **Requires decoration of outgoing messages and inspection of incoming messages**

Time Management Design Features

- Extensibility
  - provides minimal impact of LBTS algorithm modifications
  - future changes to specification
- Performance
  - efficient FIFO message delivery
  - avoids time creep
  - asynchronous LBTS algorithm avoids artificial blocking
  - scalable for large federations
  - use of a reduction network
- Robust
  - handles processor failures
  - handles delayed messages

Time Management Main Class Diagram



Initiation of LBTS Collaboration Diagram